Département d'informatique
UFR de Sciences

École doctorale SVT

Qualité de Service et calcul de chemins dans les réseaux inter-domaine et multicouches

THÈSE

présentée et soutenue publiquement le 26 septembre 2014

pour l'obtention du

Doctorat de l'université de Versailles Saint-Quentin

(spécialité informatique)

par

Mohamed Lamine Lamali

Composition du jury

Président :	Bruno Tuffin	Directeur de recherche, INRIA Rennes
Rapporteurs :	Jean-Claude Konig	Professeur, Montpellier II
	Sébastien Tixeuil	Professeur, Paris VI
Examinateurs :	Dominique Barth	Professeur, Université de Versailles-Saint-Quentin-en-Yvelines
	Johanne Cohen	Chargée de recherche CNRS
	Nicolas Le Sauze	Ingénieur de recherche, Alcatel-Lucent
	Hélia Pouyllau	Ingénieur de recherche, Thales Research & Technology
	Corinne Touati	Chargée de recherche, INRIA Grenoble
Invité :	Philippe Jacquet	Directeur de recherche, Alcatel-Lucent

Laboratoire PRISM

Cette thèse a été effectuée aux Bell Labs France d'Alcatel-lucent dans le département Advanced Internet Research (AIR, ex SAT) sous la supervision d'Hélia Pouyllau puis de Nicolas Le Sauze. Elle a été partiellement financée par la Commission Européenne via le projet de recherche FP7 ETICS.

Remerciements

Il était une fois... je soutins une thèse. Non, ça ne marche pas, il ne faut pas perdre la date de vue : j'ai soutenu ma thèse au laboratoire PR*i*SM, le 26 septembre 2014. Et l'heure ? L'heure a aussi de l'importance. D'accord : le jour. Non, il est important d'être plus... A dix heures sonnantes, exactement. Les bras de la pendule ont formé un angle de 60° pour accueillir mon manuscrit avec respect.

Naturellement, cela n'aurait jamais eu lieu sans le concours de nombreuses personnes qui m'ont dirigé, aidé, soutenu, encadré, rouspété, menacé et enfin (j'espère) félicité. Je tiens à commencer par remercier ceux qui ont joué le plus grand rôle dans la préparation de cette thèse, mes quatre encadrants (que je cite par ordre alphabétique pour éviter les susceptibilités de préséance) :

- Mon directeur de thèse, Dominique Barth, que j'ai eu d'abord la chance d'avoir comme professeur en Master 2 et dont le cours m'a fait un peu plus aimer les graphes et l'algorithmique. Je le remercie de m'avoir parlé de ce sujet de thèse et de m'avoir montré la direction à prendre quand je ne savais plus où aller. Je le remercie aussi pour sa rigueur scientifique et sa passion (contagieuse) des graphes. J'aurais aimé passer mes années de thèse à l'écouter en permanence et à profiter de son savoir et de sa culture infinis, que ce soit concernant l'isomorphisme de graphes, la Commune de Paris ou les films de Chéreau.
- Johanne Cohen, la-femme-qui-fait-des-preuves-de-NP-complétude-plus-vite-que-son-ombre (preuves qui s'avèrent très souvent correctes, en plus). Je la remercie pour son aide, son soutien et son implication dans l'encadrement de cette thèse. *But what does the brain matter compared with the heart ?* Johanne, c'est aussi la gentillesse débordante, les fous rires et les clefs perdues à chaque escapade. Merci pour tout ça.
- Nicolas Le Sauze, qui m'a récupéré après le départ d'Hélia (alors qu'il n'avait rien demandé, le pauvre). Je le remercie pour son aide, sa disponibilité, ses encouragements, ses nombreuses relectures et ses conseils avisés. Je le remercie aussi d'avoir fait en sorte que je puisse travailler dans un environnement serein et dans des conditions idéales pour effectuer cette thèse.
- Enfin, mon maître Jedi, Hélia Pouyllau. Le petit padawan que je suis lui est reconnaissant pour son encadrement rigoureux et stimulant, le temps qu'elle a accordé à cette thèse, son enseignement de la Force et de la Recherche, sa patience face à mon ignorance des acronymes réseau, et surtout sa motivation sans faille qu'elle me transmettait autant que possible, le tout presque sans recourir à la violence physique.

Si vous êtes encadrant, cherchez votre nom ici ⇒

Je remercie vivement Jean-Claude Konig et Sébastien Tixeuil d'avoir accepté de relire mon manuscrit malgré leur emploi du temps très chargé. Je remercie également les autres membres du jury : Corinne Touati [1] et Bruno Tuffin qui me fait l'honneur de présider le jury. Enfin, je suis honoré que mon nouveau chef, Philippe Jacquet, se soit laissé inviter dans ce jury.

Ici c'est le jury ⇒

Merci à Anne Bouillard de m'avoir fait découvrir les algèbres $(\max,+)$ et les galettes à la farine de châtaigne. Je lui suis redevable d'une bonne partie du chapitre 2.

Un grand merci à Karima Benatchba, dont le cours de Théorie des Langages en 3^{e} année m'a été étonnamment utile pour calculer des chemins, et à Alain Denise pour son aide et la documentation qu'il a fournie sur le même sujet.

Cette thèse ayant été financée par une bourse CIFRE, elle a été effectuée en grande partie aux Bell Labs France d'Alcatel-Lucent. J'y ai donc connu pas mal de personnes, admirables pour la plupart, que je tiens à remercier. En premier lieu, Martin Vigoureux, le chef de département le plus génial de l'Univers, pour son aide, son soutien et ses inestimables efforts pour que ma thèse se passe dans les meilleures conditions [2]. Je remercie également Richard Douville, Laurent Ciavaglia et François Dorgeuille pour leur aide précieuse en réseau, protocoles et autres joyeusetés, ainsi que tous les autres membres de l'équipe AIR (ex SAT), en particulier Ludovic Noirie, mon co-bureau, pour les discussions passionnantes sur l'histoire de France, la généalogie, la théorie des cordes, la physique quantique, et quelques fois aussi (mais rarement) l'informatique et les réseaux. Un merci également pour ceux qui nous ont quittés : Magali Prunaire, qui a tant de fois empêché mes fichiers de sombrer dans les méandres de SVN, Zied Ben Houidi, Samir Ghamri-Doudane, Walid Benchaïta et les anciens thésards qui sont partis vers un monde meilleur, en particulier Nabil Djarallah, ex co-bureau et actuel ami pour son aide et sa gentillesse, Leïla Bennacer pour les blagues que sa présence nous permettait de faire et Sameh Ben Fredj pour les bêtises que sa présence nous empêchait de dire. Un merci prospectif à tous les membres de ma nouvelle équipe

Si vous travaillez pour Bell Labs, votre nom doit être dans ce paragraphe ⇒

1. Un autre très grand merci pour l'organisation de Valuetools 2012 en Corse. C'était génial !
2. Je profite également des remerciements pour m'excuser pour toutes les cigarettes que je lui ai taxées.

avec qui le travail s'avère déjà être agréable, et un hommage à Esther Lepoix, le rayon de soleil des Bell Labs, dont la bonne humeur et les cheveux irradient les bureaux dès le matin.

Cette thèse s'est déroulée dans le cadre du projet européen ETICS. Je remercie tous les membres du projet avec qui j'ai eu des échanges très instructifs et enrichissants, en particulier Jean-Louis Rougier.

Mais, CIFRE ou pas CIFRE, pas de thèse non plus sans labo académique. Pour moi, il s'agit du vénérable PR*i*SM. Je remercie mes compagnons de fortune, embarqués à bord de la galère de la thèse, ainsi que ceux qui s'en sont sortis : Karim Bessaoud, Mariem Krichen, Isma Sadoun, Kim-Tam Huynh et ceux que j'oublie de citer. Quant à ceux qui sont encore loin de la sortie, je n'ai qu'une seule chose à dire : bon courage ! Le PR*i*SM, c'est aussi des professeurs [3], dont certains ont eu l'ingrate tâche de m'enseigner quelque chose en M2. Un merci particulier à Thierry Mautor, pour sa gentillesse et sa légendaire bonne humeur, ainsi qu'à Devan Sohier, Franck Quessette, Sandrine Vial, Jean-Michel Fourneau et tous les enseignants du PR*i*SM.

Bon, il reste la famille... Je remercie donc mes parents d'avoir corrigé les innombrables fautes d'orthographe dans ce manuscrit, ainsi que de toujours m'avoir poussé à continuer mes études quelles que soient les circonstances (je crois qu'on arrive au bout, là). Je remercie également mes sœurs (bien que je ne voie pas très bien quelle a été leur contribution à cette thèse...)

...et les amis aussi. Merci à Henda, Mehdi, Amel, Clara, Hichem, Rassim et tous ceux que je ne cite pas parce que je ne suis pas forcément en bons termes avec eux au moment où j'écris ces lignes. Je remercie spécialement Héra (alias *Le Chat* - ou *Choupette* pour les intimes) pour ses divins ronrons qui consolent bien des tracas, même ceux de la thèse.

Enfin, merci à Romain de m'avoir supporté pendant toutes ces années et de me montrer qu'il n'y a pas que les livres et les algorithmes dans la vie.

Si vous êtes membre du PR*i*SM, cherchez votre nom dans ce paragraphe ⇐

⇐ Famille

⇐ Amis et chat

3. Et des stagiaires également. Mais comme ils ne restent pas longtemps on ne se rappelle plus de leur nom au moment d'écrire les remerciements.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

L'invention des réseaux de télécommunication, et particulièrement du plus grand d'entre eux, Internet, a constitué une révolution historique qui peut être comparée à l'invention de l'écriture ou de l'imprimerie, que ce soit par son importance ou par ses conséquences. En effet, la distance ne constitue plus un obstacle à la communication, à la diffusion de la culture et du savoir et à l'échange d'idées.

Depuis, les services et les possibilités offerts par ces réseaux ont grandement évolué : du simple transport de voix (sur les réseaux téléphoniques) et de la messagerie électronique (sur Internet), ces services sont passés, en quelques décennies, à la vidéo à la demande en Haute Définition, au transfert de fichiers volumineux, à la téléprésence, aux jeux en ligne, aux transferts monétaires sécurisés, etc. On peut assister en direct et en Haute Définition à un événement ayant lieu de l'autre côté de la planète.

A l'origine, ces réseaux n'ont été ni créés ni dimensionnés pour cela. On peut dès lors se demander comment la capacité de transfert des réseaux a pu évoluer au point de supporter un volume de données aussi gigantesque. Il se trouve que des évolutions techniques très rapides ont eu lieu. Le passage du câble de cuivre à la fibre optique a considérablement augmenté le volume de données que l'on pouvait transférer, la miniaturisation des appareils électroniques a permis d'augmenter la densité des composants et ainsi d'accélérer le traitement des données. L'évolution des équipements et des moyens de transmission a pu suivre l'évolution de la quantité de données à échanger. En fait, elle a même permis de déployer des capacités supérieures à ce que cette quantité de données nécessitait, ceci afin de pouvoir gérer des pics de trafic et des augmentation pas toujours régulières ou prévisibles. Les réseaux sont surdimensionnés : la plupart du temps, seul un petit pourcentage des ressources est utilisé.

Il était très tentant de laisser les choses évoluer ainsi, l'augmentation des capacités technologiques accompagnant harmonieusement l'augmentation du trafic. Malheureusement, c'est impossible pour plusieurs raisons :

- Des raisons techniques tout d'abord : l'augmentation de la capacité des équipements ralentit, et ce n'est pas le cas de l'augmentation du trafic. Ces équipements sont soumis à de multiples contraintes qui sont difficiles à concilier : minimiser la consommation électrique, augmenter la rapidité de traitement et la densité, etc. De plus, la capacité de transmission commence à s'approcher de limites théoriques qui ne pourront pas être dépassées.
- Des raisons économiques également : maintenir le surdimensionnement des réseaux coûte cher aux opérateurs. Même si ceux-ci ne récupèrent qu'une petite partie des revenus générés par les réseaux, ils pouvaient jusqu'à présent financer le surdimensionnement grâce à l'augmentation très rapide du nombre d'utilisateurs finaux (et donc l'augmentation du marché). Or, le marché est limité à la population mondiale qui, bien que nombreuse, est finie. Par exemple, le taux de pénétration d'Internet et celui de la téléphonie mobile ont quasiment atteint le seuil maximum en Europe et en Amérique du Nord. Il n'est plus possible de compter sur l'apparition massive de nouveaux utilisateurs.

Il faut donc une solution qui permette à la fois de mieux gérer les ressources des réseaux en optimisant leur utilisation, mais aussi de fournir de nouvelles sources de revenus aux opérateurs afin de financer l'évolution technologique et le déploiement des réseaux.

La Qualité de Service

Une meilleure gestion des ressources peut se faire en accordant aux services et aux applications la quantité de ressources qui leur est nécessaire et qui correspond à leurs besoins. Il serait absurde de réserver autant de ressources pour un email que pour une vidéo en direct. Une telle gestion des ressources passe par la notion de Qualité de Service (*Quality of Service*,- QoS). La QoS est une garantie de certains paramètres techniques de transfert de données (bande passante, délai de transmission, etc.). Définir plusieurs niveaux (ou classes) de QoS permet d'accorder à chaque application ou service les ressources qui lui sont nécessaires pour fonctionner correctement, ceci en lui attribuant une des classes ou niveaux. Le prix à payer dépend de la classe de QoS et de ses paramètres.

Le déploiement de la QoS pose beaucoup de problèmes. Pour pouvoir les aborder, il faut d'abord revenir sur la nature et la structure des réseaux. Les réseaux à l'échelle planétaire, comme Internet, ou à l'échelle d'un continent, sont composés de plusieurs *domaines*. Ces domaines sont déployés dans une région donnée et sont interconnectés pour permettre la communication d'une région à l'autre. Les domaines sont gérés et administrés de manière indépendante. Certains domaines appartiennent à un même opérateur, mais la plupart sont gérés par des opérateurs différents. Le déploiement de la QoS au sein d'un même domaine ne pose pas de problème actuellement. Par exemple, la plupart des fournisseurs d'accès à Internet proposent un service de Vidéo à la Demande qui est assuré par une connexion entre l'utilisateur final et un serveur au sein du même domaine. Cette connexion bénéficie de certains mécanismes de QoS comme la priorisation du trafic ou la réservation de liens dédiés. Mais le déploiement de ces mécanismes en inter-domaine est plus complexe car il nécessite une coopération technique et économique entre les domaines. La coopération économique est difficile car aujourd'hui rien n'incite les domaines à collaborer, leurs intérêts n'étant pas forcément convergents. Sur le plan technique, chaque domaine utilise des technologies et des protocoles particuliers, qui ne sont pas forcément compatibles avec ceux des autres domaines. Le calcul de chemin ne peut donc pas se faire avec des algorithmes classiques qui ne prennent pas en compte cette l'hétérogénéité.

La plupart des travaux sur le déploiement de QoS en inter-domaine se sont focalisés soit sur l'aspect économique (concurrence et collaboration, négociation, partage des revenus etc), soit sur l'aspect technique (hétérogénéité des technologies, architecture, etc.). Le projet européen FP7 *Economics and Technologies for Inter-Carrier Services* (ETICS) aborde les deux aspects parallèlement et prend en compte les liens entre les deux (influence du modèle économique sur l'architecture et inversement). Il a pour but de proposer des solutions techniques et économiques pour le déploiement de la QoS en inter-domaine. Ces solutions passent par la définition d'un cadre économique pour la coopération entre domaines. Le projet a défini trois possibilités pour ce cadre : une *association ouverte*, une *fédération* ou une *alliance*. Chacune de ces possibilités définit un degré de coopération (du moins élevé et distribué pour l'association au plus élevé et centralisé pour l'alliance). Néanmoins, toutes les possibilités se basent sur la notion de *chemin avec qualité assurée*, qui est un chemin dans le réseau avec une garantie de QoS. Ce chemin se construit sur la base de *Service Level Agreements* (SLA), qui sont des contrats bilatéraux entre domaines. Un SLA spécifie la QoS qu'un domaine accorde à un autre pour le traverser, il a une durée déterminée et un prix. Le SLA peut également spécifier des paramètres économiques ou juridiques comme le montant et les modalités d'indemnisation si le contrat n'est pas respecté. Un SLA peut être vu comme une portion de chemin avec une certaine QoS traversant un domaine. La construction du chemin avec qualité assurée se fait en mettant bout à bout des SLA sur une suite de domaines, de la source jusqu'à la destination. Le projet ETICS a proposé des solutions centralisées et distribuées pour la construction d'un chemin avec QoS à partir de SLA. Que ce soit en centralisé ou en distribué, les domaines reçoivent une requête de QoS et doivent choisir quel SLA proposer pour la construction du chemin avec QoS. Ce processus est appelé *négociation de SLA*.

Problèmes traités et contributions

Cette thèse a été effectuée au sein du projet ETICS, elle se divise en deux parties correspondant aux aspects économique et technique de la problématique et du projet. La première partie a pour objet la négociation de SLA. Nous nous intéressons au problème rencontré par un domaine qui doit proposer un SLA : quel SLA proposer pour maximiser ses revenus à long terme ? Ce n'est pas forcément celui dont la marge de profit est la plus élevée. En effet, beaucoup de paramètres sont à prendre en compte : maximiser les chances d'être sélectionné en cas de concurrence, optimiser l'utilisation des ressources pour

les futurs SLA, minimiser la probabilité que le SLA ne soit pas respecté (violation des paramètres de QoS), etc. Le client qui envoie une requête de QoS se base également sur plusieurs paramètres pour choisir entre plusieurs propositions concurrentes : le prix qu'il veut minimiser, la QoS qu'il veut maximiser, et peut-être la réputation du domaine fournisseur de SLA.

Dans un premier temps, nous ignorons la réputation et nous proposons un modèle pour les différents acteurs de la négociation de SLA : la fonction de choix du client, la politique de prix des SLA, les ressources des domaines, etc. Il s'avère que les différents états possibles d'un domaine et les transitions entre eux forment un graphe $(\max, +)$. Maximiser les revenus d'un domaine revient à trouver le cycle de poids moyen maximum dans ce graphe. Nous montrons que la même technique permet de maximiser le revenu d'un système composé de plusieurs domaines si ceux-ci collaborent. Nous analysons ensuite le cas où les domaines sont en concurrence et montrons que le gain à l'équilibre peut être moindre que le revenu maximum calculé précédemment. En fait, le Prix de l'Anarchie peut être arbitrairement grand. Cependant, cette technique exacte permettant de trouver la stratégie optimale d'un domaine n'est possible qu'en connaissant tous les paramètres du système. Or, certains paramètres ne peuvent pas être connus par les domaines : les offres des concurrents, les ressources des concurrents, la sensibilité du ou des clients au prix ou à la QoS, etc. L'utilisation de méthodes ne nécessitant pas ces connaissances s'impose. Nous avons donc adapté des algorithmes d'apprentissage au problème de la négociation de SLA. Ces algorithmes se basent uniquement sur l'historique des négociations d'un domaine pour apprendre quel SLA proposer dans chaque situation et pour chaque type de requête. Les simulations montrent que face à un concurrent qui n'apprend pas ces algorithmes convergent très vite vers de bonnes stratégies. Mais si le concurrent utilise aussi les mêmes algorithmes, le résultat dépend fortement du type et des paramètres des algorithmes opposés. Certains convergent très vite alors que d'autres convergent plus lentement mais finissent par l'emporter.

Dans un deuxième temps, nous introduisons un mécanisme de réputation dans le modèle, ce mécanisme représente la sensibilité des clients à la réputation des domaines fournisseurs de SLA. En effet, plus les SLA proposés par un domaine échouent (c'est-à-dire que les paramètres de QoS ne sont pas respectés), plus sa réputation sera basse. Or, plus les ressources du réseau sont utilisés, plus la probabilité d'échec des SLA est grande. Vendre trop de SLA entraine une utilisation élevée des ressources du réseau, une probabilité d'échec plus élevée, et donc une réputation basse qui va défavoriser le domaine sur le long terme. Il faut trouver un compromis entre la réputation et la quantité de ressources utilisées. Nous adaptons les algorithmes d'apprentissage pour qu'ils prennent en compte ce nouveau paramètre. Les simulations effectuées montrent que ces algorithmes donnent de meilleurs résultats que des stratégies plus classiques et qu'ils trouvent un compromis entre une réputation élevée et des revenus immédiats satisfaisants.

Une fois les SLA négociés et les domaines participant au chemin identifiés, il faut *instancier* les SLA et le chemin. En effet, les SLA peuvent être définis à l'avance par les domaines (c'est-à-dire que leurs paramètres sont fixés) mais la mise en place technique du chemin (les ressources et équipements utilisés par chaque SLA à l'intérieur d'un domaine, le chemin intra-domaine utilisé, les protocoles utilisés sur chaque partie du réseau, etc.) se fait après la négociation de SLA. Le problème qui se pose ici est de gérer l'hétérogénéité des protocoles et des technologies entre les différents domaines, et parfois même à l'intérieur d'un même domaine. En effet, le chemin de bout en bout établi traversera différents domaines, et passera par des portions de réseaux utilisant différents protocoles. Or, ces protocoles sont souvent incompatibles et il n'existe pas forcément de fonction de *conversion* d'un protocole vers un autre. Une possible solution technique consiste à utiliser des fonctions d'*encapsulation* et de *désencapsulation*. Un protocole est *encapsulé* dans un autre pour traverser une portion de réseau qui n'est pas compatible avec le premier protocole. Une fois cette portion traversée, le protocole initial est *désencapsulé* avant d'atteindre la destination. Ces encapsulations peuvent être imbriquées, et intervenir à différents points du chemin. Un protocole ne peut être désencapsulé que s'il est le dernier à avoir été encapsulé, et les fonctions d'encapsulation et de désencapsulation ne sont possibles qu'à certains nœuds du réseau. Un chemin est dit *faisable* s'il respecte ces contraintes.

La deuxième partie de la thèse traite le problème de calcul de chemins faisables. Notre but est de calculer un chemin faisable avec QoS dans un contexte d'hétérogénéité en prenant en compte les fonctions d'encapsulation et de désencapsulation. Il s'est avéré que même en relaxant les contraintes de QoS (y compris la contrainte de bande passante), le problème est loin d'être trivial. En considérant l'ensemble des protocoles comme un alphabet, nous constatons que la suite de protocoles utilisés dans un chemin faisable représente un certain motif (i.e., l'ensemble de ces suites forme un langage à contexte

libre). Nous proposons de modéliser un réseau avec fonctions d'encapsulation et de désencapsulation par un automate à pile, les encapsulations correspondant à des empilements et les désencapsulations à des dépilements. Trouver le chemin faisable le plus court revient à trouver le mot le plus court accepté par l'automate. Grâce à des outils de Théorie des Langages (conversion de l'automate en grammaire, génération de mots par une grammaire, etc.) nous proposons un ensemble d'algorithmes permettant de calculer un chemin faisable minimisant le nombre de liens ou minimisant le nombre d'encapsulations. Nous prouvons la correction de ces algorithmes ainsi que leur complexité polynomiale. Notre solution est donc la première solution polynomiale au calcul de chemins faisables. En utilisant des automates à pile pondérés, nous généralisons cette solution au calcul de chemin faisable le plus court sous n'importe quelle mesure additive. Notre solution a de nombreuses applications : calcul de chemins dans les réseaux multicouches, établissement de tunnels sur Internet, etc.

Le problème de décision associé au calcul d'un chemin faisable (i.e., le problème d'existence d'un chemin faisable) est NP-complet si on ajoute une contrainte de bande passante. Nous améliorons ce résultat en montrant que le problème reste NP-complet même s'il n'y a que deux protocoles. Enfin nous adaptons un algorithme de calcul de chemins avec QoS pour qu'il prenne en compte les fonctions d'encapsulation et de désencapsulation.

Organisation de la thèse

Cette thèse se compose de deux parties et de cinq chapitres :

- **Le chapitre 1** définit d'abord la QoS et sa mise en place au sein d'un même domaine. Le routage inter-domaine actuel, le système économique d'Internet et la relation entre les deux sont présentés. Nous discutons ensuite la possibilité d'étendre le protocole de routage inter-domaine actuel à la QoS. Nous présentons le projet ETICS et quelques unes des solutions qu'il propose et plaçons nos contributions dans ce contexte.
- **La première partie** de la thèse comprend les chapitres 2 et 3 et aborde le problème de négociation de SLA :
  - **Le chapitre 2** traite le problème de négociation de SLA sans prise en compte de la réputation. Notre solution exacte basée sur les algèbres $(\max, +)$ est présentée dans le cas d'un domaine fournisseur de SLA en situation de monopole et dans le cas de deux domaines fournisseurs qui collaborent. Nous étudions le même système quand les domaines sont égoïstes et calculons ainsi le Prix de l'Anarchie. Nous discutons ensuite des limitations de cette méthode. Différents algorithmes d'apprentissage sont présentés ainsi que leurs caractéristiques (vitesse de convergence, complexité, etc.). Nous adaptons ces algorithmes au problème de négociation de SLA et exposons leur comportement dans un cas où il n'y a pas d'ordre de préférence sur les SLA, grâce à des simulations.
  - Dans **le chapitre 3**, nous introduisons un mécanisme de réputation sur les domaines et observons son influence sur le problème de négociation de SLA. Nous adaptons les algorithmes d'apprentissage présentés au chapitre précédent pour qu'ils prennent en compte ce nouveau paramètre. Enfin, nous présentons des résultats de simulations montrant que ces algorithmes s'adaptent à la modification du problème et trouvent un compromis entre une réputation basse et un usage excessif des ressources du réseau.
- **La deuxième partie** de la thèse comprend les chapitre 4 et 5 et traite de la prise en compte des encapsulations de protocoles dans le calcul de chemins :
  - **Le chapitre 4** présente d'abord la structure en couches des réseaux ainsi que l'encapsulation de protocoles de couches hautes dans des protocoles de couches basses. L'architecture Pseudo-Wire, qui permet l'encapsulation de protocoles de couches basses dans des protocoles de couches plus hautes, est ensuite présentée. Enfin, nous présentons plusieurs cas de figure où le problème de calcul de chemins avec des encapsulations se pose.
  - **Le chapitre 5** définit ce même problème de manière formelle et explique sa relation avec la Théorie des Langages. Une solution pour calculer le chemin le plus court en nombre de liens ou d'encapsulations est présentée. Nous donnons les algorithmes utilisés, leur complexité et ainsi que la preuve de leur correction. Une généralisation pour calculer le chemin le plus court sous n'importe quelle mesure additive est aussi proposée. Nous donnons ensuite la preuve que le problème est NP-complet sous contrainte de bande passante, même avec un ensemble restreint

de deux protocoles. Nous présentons brièvement l'algorithme SAMCRA [4] qui calcule des chemins sous contraintes de QoS et l'adaptons afin qu'il prenne en compte les encapsulations de protocoles.

Nous terminons par une conclusion générale où nous résumons nos contributions et explorons quelques perspectives de recherche. Cette thèse a donné lieu à une publication dans un journal international, quatre publications dans des conférences internationales et deux publications dans des workshops internationaux. Elle a également donné lieu à plusieurs présentations : Bell Labs Open Days (journées portes ouvertes des Bell Labs avec présentations et démonstrations des travaux en cours), journées ResCom (présentation de travaux sur les réseaux), workshop industriel ETICS, etc., ainsi qu'à trois brevets. Les publications sont listées en page 105.

4. SAMCRA : Self-Adaptive Multiple Constraints Routing Algorithm.

1

La Qualité de Service dans les réseaux inter-domaine

1.1 Introduction

L'usage actuel des services et applications en télécommunication impose de plus en plus de contraintes aux réseaux. En effet, ces applications sont très gourmandes en bande passante (comme par exemple la Vidéo HD), mais peuvent nécessiter également des délais de transmission très courts (diffusion d'événements en direct), une connexion sécurisée, etc. Ces applications ne peuvent pas s'étendre et se développer s'il n'y a pas de garantie sur ces paramètres.

On appelle Qualité de Service (*Quality of Service*,- QoS) la spécification de ces paramètres à certaines valeurs : garantie de ne pas descendre sous une valeur seuil de bande passante, garantie de ne pas dépasser un délai de transmission d'une certaine durée, etc. Les applications nécessitant une garantie de QoS sont nombreuses et ont tendance à se généraliser avec l'orientation des réseaux sur le contenu. Par exemple, un service de chirurgie à distance ne peut pas être envisagé sans une garantie de QoS stricte. Et l'augmentation exponentielle du trafic sur les réseaux rend de plus en plus difficile la garantie de ces paramètres sans mécanismes dédiés.

Le déploiement de mécanismes pour garantir la QoS peut se faire à l'intérieur d'un même domaine. Ces mécanismes peuvent se baser sur une priorisation du trafic ou la réservation de certains liens et équipements à un service donné. Mais le déploiement de la QoS sur un réseau inter-domaine, comme Internet, est beaucoup plus complexe. Le protocole de routage inter-domaine actuel n'a pas été conçu pour assurer la QoS, et sa modification pour la supporter est difficile. De plus, les mécanismes envisagés pour assurer la QoS se heurtent à des difficultés dues à la nature même des réseaux inter-domaine : les domaines ne collaborent pas et chacun a pour but de maximiser son profit. Or, la collaboration est indispensable pour la mise en place d'une solution cohérente.

Le projet européen FP7 *Economics and Technologies for Inter-Carrier Services* (ETICS), qui est le cadre de cette thèse, propose des solutions basées sur la négociation de contrats, appelés *Service Level Agreements* (SLA), entre les domaines. Ces contrats spécifient une certaine QoS qu'un domaine assure à un autre en échange de paiement. Les domaines doivent s'engager dans des associations ou des alliances permettant d'échanger des informations, et donc de collaborer, jusqu'à un certain point pour associer leurs SLA et former des routes inter-domaine garantissant la QoS. Plusieurs solutions sont proposées selon la nature des informations échangées et du degré de collaboration entre domaines. Les contributions de cette thèse, qui portent sur la négociation de SLA et sur le calcul de chemins dans les réseaux inter-domaine, trouvent leur contexte dans les solutions proposées par ETICS.

Ce chapitre se présente comme une introduction à la QoS dans les réseaux, et plus généralement à cette thèse. La section 1.2.1 définit la QoS et évoque les besoins qu'en ont les applications réseaux. La section 1.3 présente le déploiement de la QoS au sein d'un même domaine et cite certains mécanismes utilisés. La section 1.4 présente les réseaux inter-domaine et particulièrement Internet, elle explique le système économique qui le régit et le protocole de routage qui y est utilisé actuellement. Les difficultés du déploiement de la QoS en inter-domaine et les solutions possibles sont également développées. La section 1.5 présente le projet ETICS et les solutions qu'il propose. Le positionnement du projet par rapport à la Net neutralité est brièvement abordé, bien que cette discussion déborde largement du cadre de cette thèse. La section 1.6 présente les problèmes traités dans cette thèse et les contributions apportées.

Enfin, la section 1.7 conclut le chapitre.

1.2 La Qualité de Service

1.2.1 Définition de la Qualité de Service

La QoS est représentée par un ensemble de paramètres techniques à garantir durant une connexion. Comme exemples de ces paramètres, on peut citer :

- **La bande passante (ou débit binaire) :** C'est la quantité de données pouvant être transmise durant un laps de temps. Le plus souvent, elle est mesurée en nombre de *bits/seconde* (bits par seconde,- bps) ou un de ses ordres de grandeur : kilobits par seconde ou Kbps (10^3 bps), mégabits par seconde ou Mbps (10^3 Kbps), gigabits par seconde ou Gbps (10^3 Mbps), etc. La bande passante est un paramètre clé de la vitesse de transmission des données : plus la bande passante est élevée, plus la transmission d'un fichier sera rapide. Ce paramètre est également crucial pour les applications temps réel. Par exemple, la vidéo en direct nécessite une bande passante plancher sous laquelle on ne doit pas descendre pour garantir une transmission en temps réel des images. La contrainte de bande passante est donc une valeur seuil de bande passante disponible sous laquelle il ne faut pas descendre.
- **Le délai (ou latence) :** C'est le temps qui sépare l'émission d'un paquet de données de l'arrivée à sa destination. Il est mesuré en secondes ou un de ses ordres de grandeur (de quelques milisecondes pour les transmissions nécessitant une très grande réactivité à plusieurs minutes pour les applications tolérantes aux délais). Une contrainte de délai définit une temps maximum à ne pas dépasser pour tous les paquets. Les applications temps réel et interactives sont très sensibles au délai. Par exemple un jeu vidéo en ligne (*gaming* en anglais) nécessite un délai très court (non perceptible par les joueurs) pour garantir la fluidité de l'interaction entre les joueurs et avec le jeu.
- **Le taux de perte :** Ce paramètre représente le pourcentage de paquets de données perdus lors d'une transmission. Dans une transmission orientée connexion, la perte d'un paquet occasionne sa réémission, ce qui augmente donc le délai moyen. Dans une transmission sans connexion, le paquet perdu n'est pas forcément réémis ce qui occasionne une perte d'information. Un taux de perte élevé peut augmenter le temps de transmission d'un fichier ou dégrader la qualité d'une transmission vidéo. La contrainte de taux de perte définit un seuil à ne pas dépasser pour garantir la QoS.

Ces trois paramètres seront pris en exemple tout le long de cette thèse pour illustrer les différents problèmes et algorithmes concernant la QoS. La raison en est que, comme nous le verrons plus loin, elles représentent les trois types de contraintes généralement rencontrées : contraintes concaves (bande passante), contraintes additives (délai), et contraintes multiplicatives (taux de perte). Néanmoins, il existe plusieurs autres paramètres de QoS qui peuvent être contraints : la gigue (variation du délai), la disponibilité, certains niveaux de sécurité, etc.

1.2.2 Les besoins en Qualité de Service

L'augmentation spectaculaire du trafic sur Internet ainsi que l'émergence de nouvelles applications et nouveaux services rendent le besoin de QoS de plus en plus évident. Parmi ces services, on peut citer :

- **La vidéo :** Que ce soit sous forme de vidéo à la demande (*Video on Demand*,- VoD), de visioconférence, de streaming, etc., la vidéo impose une contrainte de bande passante minimum pour que l'image soit de bonne qualité, d'autant plus depuis l'apparition de la Haute Définition (HD). La fluidité des images dépend fortement de la gigue. De plus, si la vidéo est en direct, une contrainte de délai court va s'ajouter aux contraintes précédentes var la vidéo n'est pas « bufferisée ». L'explosion du trafic vidéo (66% du trafic total en 2013 [1]) rend difficile la satisfaction de ces contraintes pour tous les utilisateurs.
- **La voix :** La voix a longtemps été transportée sur les réseaux téléphoniques dédiés et dimensionnés en fonction des besoins de ce type de trafic. Néanmoins, la Voix sur IP [5] (VoIP) s'impose et est de plus en plus utilisée par les opérateurs téléphoniques. Certaines applications sont directement proposées sur Internet (Skype, Google Voice, etc.). Or, le réseau Internet n'a pas été conçu à la base pour transporter de la voix, la VoIP impose donc des contraintes supplémentaires au réseau :

5. IP : Internet Protocol.

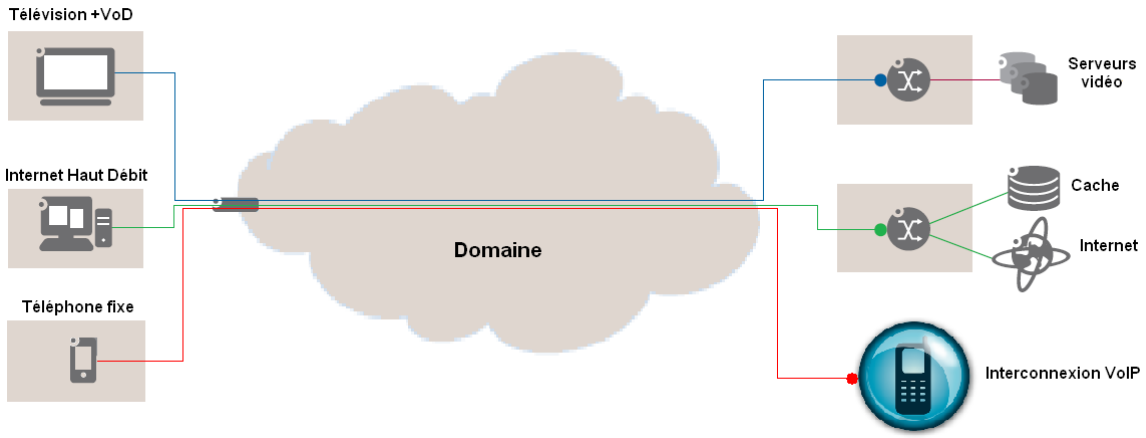


FIGURE 1.1 – Architecture Triple Play simplifiée [15].

la bande passante nécessaire est relativement faible (jusqu'à 64kbps pour une conversation humaine actuellement), mais les contraintes sur le délai et la gigue sont très strictes pour permettre la fluidité d'une conversation et minimiser le décalage entre l'émission et la réception de la voix.

– **Applications fortement interactives :** Certaines applications se caractérisent par une interactivité très forte avec l'utilisateur. On peut citer les jeux en ligne, la téléprésence, mais aussi les services médicaux à distance pour lesquels le respect des contraintes de QoS est vital pour les patients. Pour ce genre d'applications, la bande passante, et surtout le délai et la gigue, sont cruciaux pour le bon fonctionnement.

D'autres catégories de services et d'applications peuvent également être mentionnées : les transactions financières pour lesquelles des contraintes strictes de cryptage et de délais sont nécessaires, la télésurveillance dont la fiabilité nécessite un taux de perte de paquets faible, etc.

L'augmentation de l'utilisation de ces services induit une forte demande en QoS et nécessite de nouvelles solutions techniques et économiques pour les satisfaire.

1.3 La Qualité de Service en intra-domaine

Internet est composé de plusieurs réseaux inter-connectés, le plus souvent appartenant à des opérateurs différents, qu'on appelle domaines. Un domaine est donc un réseau géré par une entité unique.

Il existe des services qui sont déployés en intra-domaine uniquement, c'est-à-dire que le chemin entre le fournisseur de service et le client ne passe que par un seul domaine. Dans ce contexte, il est possible de déployer la QoS pour différentes raisons, les deux principales étant :

– L'opérateur qui gère le domaine a une vision globale de son réseau. Il connait l'état des ressources, la topologie, les équipements, les éventuelles pannes, etc.

– La prise de décision se fait de manière centralisée, il n'y a aucun besoin de coopération entre des entités indépendantes. De plus, aucun problème de compétition ou de concurrence ne se pose.

Le déploiement de la QoS en intra-domaine est donc un problème purement technique. Un exemple de déploiement de services avec différentes qualités (dont certains avec garantie de QoS) en intra-domaine est le Triple Play.

1.3.1 Exemple de services déployés en intra-domaine : le Triple Play

Le Triple Play est l'offre commerciale d'accès à Internet la plus répandue en Europe. Elle consiste, pour un fournisseur d'accès, à offrir un forfait avec la télévision (souvent avec la VoD en hors forfait), la téléphonie fixe et l'accès Internet haut débit.

La figure 1.1 montre l'architecture simplifiée sur laquelle sont déployés les services Triple Play. Le transport de la vidéo, de la VoIP et des paquets Internet n'est pas effectué de la même manière.

Les paquets destinés à ou provenant d'Internet sont traités avec la politique du *best effort*. Cette politique consiste à acheminer les paquets « au mieux » à leur destination, sans aucune garantie de QoS,

ni même aucune garantie d'arrivée des paquets à destination. Internet fonctionne actuellement sur ce principe.

Les données vidéo de la télévision ou de la VoD (qui sont des services spécialisés mono-opérateur) sont acheminées des serveurs vidéos (hébergés par l'opérateur) jusqu'à l'utilisateur final avec certains mécanismes permettant le respect de la QoS. Un de ces mécanismes est la priorité sur les paquets Internet. Cela leur permet d'éviter les délais et les pertes en cas de congestion. Les priorités mises en place ont pour but d'optimiser certains paramètres de QoS importants pour la vidéo (délai, gigue, etc.). Un autre mécanisme peut-être l'utilisation de technologies spécifiques ou de chemins dédiés (câble optique réservé à la VoD, par exemple) permettant le respect des contraintes de QoS.

Les données de VoIP sont également acheminées avec une priorité ou des technologies spécifiques permettant le respect des contraintes de QoS relatives au transport de la voix.

Le Triple Play est donc un exemple de différenciation de services, où les paquets ne sont pas traités de la même manière en fonction de leur nature.

1.3.2 Exemple de mécanisme de Qualité de Service : *DiffServ*

DiffServ (pour *Differentiated Services* : Services Différenciés) [86, 22] est une architecture réseau qui a pour but de garantir une certaine QoS pour certains types de trafic. Pour cela, un ensemble de *classes* de paquets est défini et un ordre de priorité est défini sur ces classes. A chaque classe correspondra donc un traitement spécifique appelée Traitement Par Saut (*Per Hop Behavior*,- PHB).

L'architecture *DiffServ* propose d'utiliser le champ *Type de service* dans l'en-tête des paquets IP [6] pour identifier les différentes classes. Ainsi, les 6 premiers bits de ce champ correspondront au Code de Service Différencié (*Differentiated Services CodePoint*,- DSCP), qui contiendra l'identifiant de la classe à laquelle appartient le paquet [86]. Les 2 derniers bits ont été affectés ultérieurement à un mécanisme de notification de congestion (*Explicit Congestion Notification*,- ECN) [95].

En théorie, on peut définir jusqu'à $2^6 = 64$ classes différentes de paquets. En pratique, les trois classes suivantes sont les plus utilisées :

- **Default PHB :** C'est la classe dont le traitement par défaut correspond au *Best Effort*. Tous les paquets ne remplissant pas les critères d'affectation aux autres classes sont automatiquement classés dans celle-ci. Ils sont traités sans aucun mécanisme particulier pour assurer la QoS. Les paquets de cette classe sont traités avec la priorité la plus basse.
- **Expedited Forwarding (EF) PHB [34] :** C'est une classe regroupant les paquets nécessitant un délai, un taux de perte et une gigue peu élevés, ce qui la rend adéquate pour le transport de la voix, de la vidéo et les données des applications temps réel. Les paquets de cette classe sont traités avec une priorité absolue par rapport à ceux des autres classes. Pour éviter une congestion uniquement due aux paquets de cette classe, l'admission à la classe est stricte et le pourcentage de paquets admis ne dépasse pas 30% du total des paquets.
- **Assured Forwarding (AF) PHB [55, 54] :** Il s'agit en fait d'un groupe de classes correspondant à un PHB qui minimise le risque de perte des paquets. Les paquets n'ont donc pas de traitement spécifique pour minimiser le délai ou maximiser la bande passante, mais ils seront traités avec priorité afin de minimiser le risque que ces paquets soient supprimés. Ce groupe de classes peut également contenir des priorités différentes selon les différentes classes.

D'autres classes moins utilisées sont également définies. On peut citer la classe *Voice Admit* (VA) [20] qui généralise la classe EF, *Class Selector* (CS) qui assure la compatibilité avec l'ancienne utilisation du champ *Type de service*, etc.

L'architecture de *DiffServ* se compose de quatre fonctions séquentielles appliquées à chaque paquet :

1. **Classification :** A leur arrivée, les paquets sont affectés aux classes prédéfinies. Il y a deux modes de classification : le premier se base uniquement sur le contenu du DSCP pour déterminer la classe, le deuxième peut utiliser tous les champs de l'en-tête IP pour déterminer la classe, particulièrement les champs contenant l'adresse source et l'adresse destination du paquet. Ainsi, il est possible d'implémenter un traitement différent selon la provenance ou la destination du paquet.
2. **Conditionnement :** Une fois les paquets classés, ils sont positionnés dans différentes files d'attente correspondant à leur classe.

6. La structure des paquets IP et de leur en-tête sera détaillée dans le chapitre 4 dans le but d'expliquer les mécanismes d'encapsulation.

3. **Ordonnancement :** C'est la fonction clé de *DiffServ*. L'ordonnanceur choisit à chaque instant une file et retire le premier paquet de cette file pour le traiter. Le choix de l'ordonnanceur se fait selon les règles de priorité préétablies entre les différentes classes. Ces règles ne sont pas forcément strictes : une politique de *fair queuing* (files d'attente équitables) ou *weighted fair queuing* (files d'attente équitables pondérées) peut être mise en place. La première permet d'attribuer un temps de traitement équitable aux différentes files, la seconde permet une pondération du temps de traitement (par exemple : 80% de temps pour la file 1, 20% de temps pour la file 2).
4. **Envoi :** Les paquets sont traités et envoyés dans l'ordre déterminé par l'ordonnanceur.

Comme on peut facilement le déduire de son fonctionnement et de son architecture, *DiffServ* ne permet d'assurer qu'une QoS relative. En effet, si les ressources du réseau sont largement supérieures aux besoins, il n'y aura pas de congestion et la QoS sera assurée. Autrement, *DiffServ* permet de s'assurer que les paquets de certaines classes seront plus vite traités que d'autres, ou que la bande passante allouée à certaine classe est plus grande que les autres, mais il est impossible de garantir qu'une contrainte stricte de QoS (par exemple délai $\leq$ 5ms) soit respectée. Tout ce qui peut-être assuré est que, par exemple, les paquets VoIP auront un délai inférieur aux paquets *best effort*, sans pouvoir obtenir une borne fixe sur ce délai. Les performances de *DiffServ* dépendent fortement de la congestion ainsi que du type de trafic (si la plus grande partie du trafic appartient à la même classe prioritaire, les règles de priorité n'auront aucune utilité). De plus, comme nous le verrons dans la section suivante, *DiffServ* pose des problèmes supplémentaires en inter-domaine car il nécessite une définition commune des classes et des règles de priorité pour être performant. Une coordination entre les domaines est donc indispensable à son bon fonctionnement.

1.4 Qualité de service dans un réseau inter-domaine

Bien que limité et non exempt de difficultés, le déploiement de la QoS au sein d'un même domaine est faisable et souvent efficace. Il en est tout autre pour l'inter-domaine. En effet, un tel déploiement nécessite une collaboration importante entre les différents domaines, et ce sur deux plans : le plan technique et le plan économique. La collaboration sur le plan technique est difficile (hétérogénéité des technologies et des protocoles, etc.). La collaboration sur le plan économique est encore plus complexe car elle est contradictoire avec la situation de concurrence dans laquelle se trouvent ces domaines. De plus, le modèle économique actuel régissant les relations entre domaines n'incite pas à la collaboration. Le déploiement de la QoS en inter-domaine nécessite la conception d'un autre modèle économique qui incite plus à la collaboration.

Dans cette section, nous définirons et présenterons les réseaux inter-domaine, ainsi que le protocole de routage et le modèle économique actuel. Nous montrerons les difficultés de déploiement de la QoS dans un contexte inter-domaine, que ce soit en adaptant le protocole actuel ou en utilisant de nouveaux protocoles.

1.4.1 Notions de base

- Un domaine, également appelé Système Autonome (*Autonomous System*,- AS) est une portion du réseau géré par une entité unique (opérateur, fournisseur de services, etc.) et disposant d'une politique de routage interne unique et d'une politique de routage externe cohérente. Il est administrativement défini par un Numéro de Système Autonome (*Autonomous System Number*,- ASN). Certains opérateurs ne disposent que d'un seul domaine, alors que d'autres, plus grands, en possèdent plusieurs (disséminés ou regroupés géographiquement).
- Un réseau inter-domaine est un ensemble de domaines inter-connectés disposant de moyens de routage qui permettent l'acheminement des paquets et leur transport dans le réseau. Internet est le plus grand réseau inter-domaine au monde. Il contient en avril 2014 plus de 45000 domaines [5, 6].
- Dans un contexte inter-domaine, le routage se fait en deux étapes. D'abord, les paquets sont acheminés jusqu'au domaine où se trouve la destination, possiblement en traversant plusieurs autres domaines. Une fois le domaine destination atteint, un autre mécanisme de routage permet d'acheminer les paquets à l'intérieur du domaine jusqu'à la destination. La première étape est ce qu'on appelle un routage inter-domaine, la seconde étape est un routage intra-domaine. Le routage intra-domaine

est effectué par des protocoles dédiés : OSPF [7] [84], IS-IS [8] [89], etc. Le routage inter-domaine est détaillé en section 1.4.4. Dans la suite de ce chapitre, nous nous intéresserons principalement au routage inter-domaine et la question de la QoS dans ce contexte.

1.4.2 Relations économiques entre les domaines

Actuellement, il existe deux types de relation entre deux domaines connectés :

- **Les accords de *peering* :** C'est un accord d'égal à égal entre deux domaines. Il a lieu entre deux domaines de taille comparable qui échangent un trafic plus ou moins symétrique. Le principe de l'accord est l'échange sans paiement de trafic ainsi que le partage des frais de maintenance de l'interconnexion.
- **Les accords Client-Fournisseur :** C'est un contrat entre un domaine client et un domaine qui lui fournit un service de transit (connectivité à toutes ou une partie des adresses Internet). Le domaine fournisseur facture le trafic reçu par le domaine client à celui-ci. Cette facturation peut se faire par un prix unitaire (paiement en fonction du trafic envoyé) ou au forfait. Un domaine fournisseur peut avoir (et a souvent) plusieurs clients. Un domaine client peut également souscrire à plusieurs domaines fournisseurs.

1.4.3 Structure hiérarchique d'Internet

Les deux types de relation entre les domaines ont structuré Internet de façon hiérarchique et créé trois types de domaines :

- **Les domaines Tier-1 :** Un domaine est dans cette catégorie s'il n'est client d'aucun autre domaine dans le réseau, c'est-à-dire que tous les domaines auquel il est connecté sont soit ses clients, soit des *égaux* (ils ont un lien de *peering* avec lui). Ce sont les domaines en haut de la hiérarchie d'Internet et ils sont tous directement connectés les uns aux autres par des liens de *peering*. Ils peuvent ainsi atteindre n'importe quelle partie d'Internet sans avoir recours à un transit payant. Il en existe une dizaine, parmi lesquels : LEVEL3, AT&T, etc [6].
- **Les domaines Tier-2 :** Ce sont les domaines qui ont à la fois des domaines clients et des domaines fournisseurs. Typiquement, un domaine Tier-2 est client d'un ou de plusieurs domaines Tier-1 et le fournisseur d'un ou de plusieurs domaines Tier-3. Il peut également avoir des relations de *peering*. Ces domaines ont généralement pour fonction d'assurer le transit d'autres domaines. Comme exemples, on peut citer : Tele2 (client de Sprint), British Telecom (client de NTT et LEVEL3), etc.
- **Les domaines Tiers-3 :** Ce sont les domaines qui ne sont que clients des autres domaines. Ils n'assurent jamais le transit pour d'autres domaines et n'ont pas de liens de *peering*. Ce sont généralement des fournisseurs de services (fournisseurs d'accès, etc.).

La figure 1.2 illustre la structure hiérarchique d'Internet. Les domaines Tier-1 ont des liens de *peering* entre eux. Les domaines Tier-2 sont clients des domaines Tier-1 et fournisseurs des domaines Tier-3. Enfin, les domaines Tier-3 ne sont que des clients des autres domaines.

Remarque. Il se peut que certains domaines Tier-2 aient des liens de *peering* avec certains domaines Tier-1, et soient clients d'autres domaines Tier-1. Mais ils restent Tier-2 tant qu'ils sont clients d'au moins un domaine. Par souci de simplification, nous n'avons pas illustré ce cas sur la figure 1.2.

1.4.4 Routage inter-domaine actuel : BGP

1.4.4.1 Principes de base

BGP [9] [97] est le protocole de routage inter-domaine utilisé actuellement sur Internet. C'est un protocole à vecteur de chemins (*Path Vector Protocol*), c'est-à-dire que les routeurs s'échangent des messages contenant des destinations ainsi que la route à suivre pour atteindre ces destinations. Les routes sont ainsi propagées dans le réseau, et bien qu'aucun routeur n'ait une vision globale du réseau, chacun sait quel est le prochain domaine pour atteindre une certaine destination.

7. OSPF : Open Shortest Path First.
8. IS-IS : Intermediate System to Intermediate System.
9. BGP : Border Gateway Protocol.

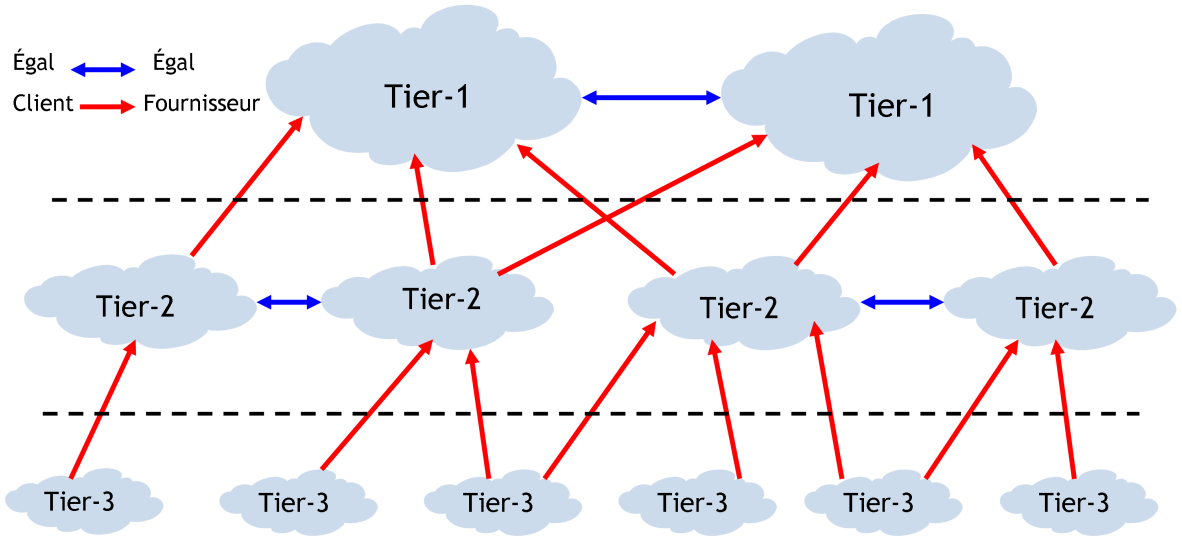


FIGURE 1.2 – Illustration de la structure hiérarchique d'Internet.

Pour des raisons de scalabilité, les adresses sur Internet sont regroupées en *préfixes* [46]. Un préfixe est un groupe d'adresses ayant le même début (sur un certain nombre de bits spécifié). Ainsi, au lieu de maintenir un chemin pour chaque adresse sur Internet (ce qui serait totalement impossible vu le nombre d'adresses existantes), les routeurs maintiennent des chemins vers les préfixes, qui sont beaucoup moins nombreux. On compte environ 500000 préfixes sur Internet en avril 2014 [5].

Pour chaque préfixe, le routeur doit disposer d'informations de base : l'adresse du prochain routeur pour atteindre cette destination, la liste des domaines à traverser pour l'atteindre (cette liste constitue la route, appelée aussi *AS Path*), l'origine de la route et quelques informations permettant de faire un choix si plusieurs routes sont disponibles (préférence locale, communauté, etc.). La *table de routage* contient ces informations pour chaque préfixe.

La construction de la table de routage se fait comme suit :

- A l'initialisation, le routeur échange des messages avec ses voisins directs et insère donc les chemins vers ceux-ci dans sa table.
- Le routeur reçoit des routes vers des préfixes de ses routeurs voisins. Il remplit ainsi sa table de routage et la maintient à jour grâce aux informations qu'il reçoit.

La figure 1.3 illustre la manière dont des routes BGP sont propagées. La figure montre un ensemble de 5 domaines interconnectés. Les routes pour atteindre le domaine E sont propagées comme suit :

1. Le routeur $e1$ du domaine E envoie le message (1) au routeur $b2$ du domaine B. Le message signifie à $b2$ que le routeur $e1$ peut atteindre le domaine E via la route E (le routeur $e1$ faisant partie du domaine E, la route est constituée de ce seul domaine),
2. Le routeur $b2$ du domaine B informe via le message (2) tous les routeurs de son domaine, y compris $b1$, qu'il connaît une route vers le domaine E,
3. Le routeur $b1$ envoie le message (3) au routeur $a1$ du domaine A pour l'informer qu'il connaît une route vers le domaine E. Cette route est : BE,
4. Le routeur $e2$ du domaine E envoie le message (4) au routeur $d2$ du domaine D. Le message signifie à $d2$ que le routeur $e2$ peut atteindre le domaine E via la route E,
5. Le routeur $d2$ du domaine D informe via le message (5) tous les routeurs de son domaine, y compris $d1$, qu'il connaît une route vers le domaine E,
6. Le routeur $d1$ envoie le message (6) au routeur $c2$ du domaine C pour l'informer qu'il connaît une route vers le domaine E. Cette route est : DE,
7. Le routeur $c2$ du domaine C informe via le message (7) tous les routeurs de son domaine, y compris $c1$, qu'il connaît une route vers le domaine E,
8. Le routeur $c1$ envoie le message (8) au routeur $a1$ du domaine A pour l'informer qu'il connaît une route vers le domaine E. Cette route est : CDE. Ainsi le routeur $a1$ aura reçu deux routes

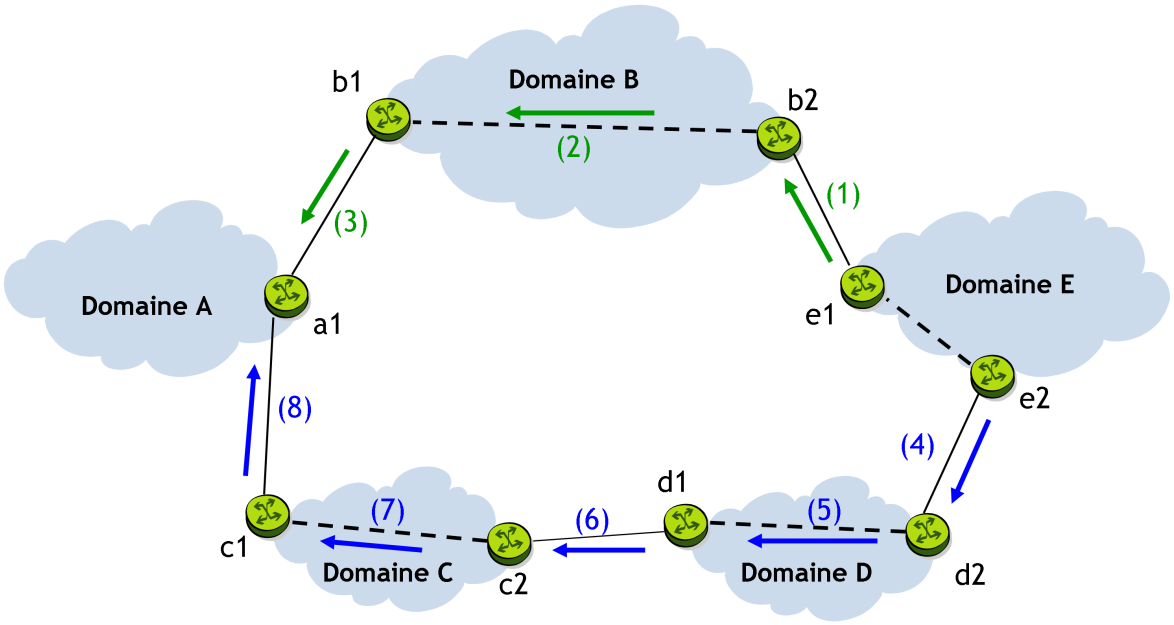


FIGURE 1.3 – Exemple de propagation de routes BGP.

distinctes vers le domaine E. Il doit en choisir une seule car il ne peut pas avoir plusieurs routes vers le même préfixe dans sa table de routage.

BGP possède des mécanismes pour détecter et supprimer les boucles (une route passant plus d'une fois par le même domaine). Ainsi, si un domaine reçoit une route dans laquelle il figure déjà, il la supprime et envoie un message d'erreur.

1.4.4.2 Sélection et filtrage de routes

Toutes les routes reçues par un routeur BGP ne sont pas forcément utilisées, ni même examinées. Il y a deux processus de filtrage pour chaque routeur : il filtre les routes reçues pour sélectionner celles qu'il va examiner, puis il filtre celles-ci pour savoir lesquelles envoyer (ou propager) à ses voisins. Entre ces deux processus, il doit comparer les routes à examiner avec les routes déjà disponibles pour n'en garder qu'une seule (par préfixe) dans sa table de routage.

Le premier processus se fait selon des règles fixées par le domaine et qui ne font pas partie de BGP. Un domaine A peut ne pas examiner les routes qu'il reçoit d'un domaine B, pour des raisons commerciales ou politiques.

Si une route est examinée puis sélectionnée et ajoutée à la table de routage, le routeur doit décider s'il la propage et à qui. Un domaine propagera les routes sélectionnées à ses domaines clients puisque ceux-ci le paient pour avoir accès à tout le réseau. Cependant un client ne propagera jamais à un de ses fournisseurs une route reçue par un autre de ses fournisseurs, car il n'a aucun intérêt économique à assurer le transit entre ses fournisseurs. Sur le même principe, un domaine ne propage pas des routes reçues par un *peer* à d'autres *peers* car il n'a aucun intérêt à assurer le transit entre ses *peers*.

1.4.4.3 Stabilité de BGP et routage *Valley-Free*

Comme on l'a vu dans la section précédente, les relations économiques entre les domaines influencent fortement leurs politiques de routage. Un domaine assure le transit entre un de ses clients et un de ses fournisseurs, mais il n'assurera pas le transit (et donc ne propagera pas les routes) entre deux de ses fournisseurs ou deux de ses *peers*. Ces politiques ont des conséquences sur certaines caractéristiques de BGP. Nous définirons d'abord ces caractéristiques avant de voir l'influence que les politiques de routage exercent sur eux.

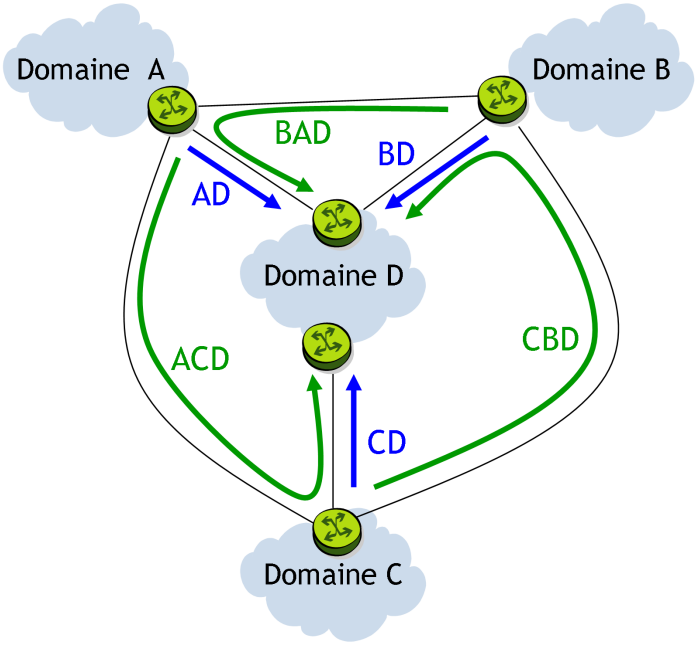


FIGURE 1.4 – Exemple de configuration où le routage ne peut être stable. Les domaines A, B, et C préfèrent les routes en vert aux routes en bleu pour atteindre le domaine D [52].

Stabilité et convergence : On dit que le routage est dans un état *stable* si, en l'absence de tout changement dans le réseau (panne, nouveau lien, changement de politique, etc.), les tables de routages (et donc les routes) ne changent plus et le routage se fait correctement. Un protocole de routage est dit *convergent* si en partant de n'importe quel état, et en l'absence de changements dans le réseau, le routage atteint un état stable en un temps fini.

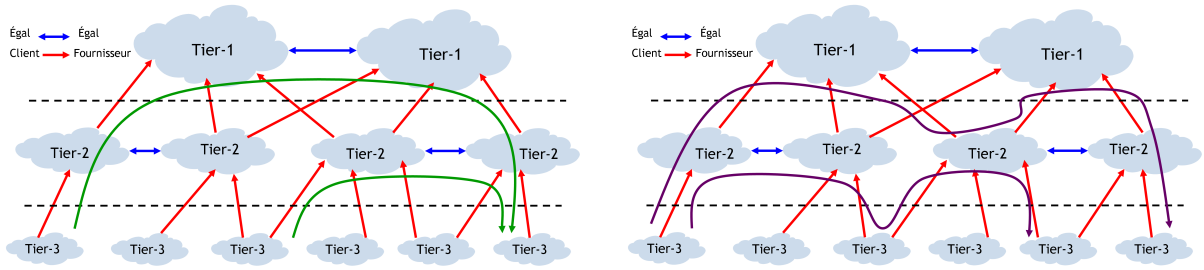
Oscillations de routes : De prime abord, il n'est pas évident que BGP soit stable. Une illustration de l'instabilité est le phénomène *d'oscillation de routes*. C'est ce qui se produit quand il y a un cycle dans la mise à jour des routes et que les tables de routages sont modifiées continuellement. La figure 1.4 montre un réseau constitué de quatre domaines. Chaque domaine préfère la route en vert plutôt que la route directe en bleu pour atteindre le domaine D. Par exemple, le domaine A préfère la route ACD à la route AD. Si on implémente BGP avec ces politiques de routage sur le réseau de la figure 1.4, les routes ne se stabiliseront jamais. Au début, chaque domaine connaitra uniquement la route directe vers D. Puis, le domaine A recevra la route CD du domaine B et effacera donc la route directe AD. Mais quand le domaine C recevra la route BD ; il effacera la route directe CD et il notifiera cette suppression au domaine A. Celui-ci rétablira la route AD et le notifiera au domaine B, qui à son tour supprimera la route BD et en informera le domaine C, qui rétablira la route CD, etc. Il y a un cycle dans la mise à jour des tables de routage qui fait qu'il n'existe aucun état stable respectant les préférences des domaines. Cet exemple est issu des travaux de Griffin et Wilfong sur l'instabilité du routage inter-domaine [52].

Il se trouve que les relations économiques entre domaines et la structure hiérarchique d'Internet permettent d'éviter l'instabilité de BGP. Gao et Rexford [48] ont montré que des règles locales simples de sélection de routes garantissent non seulement l'existence d'un état stable de routage, mais également la convergence de BGP vers cet état stable. Ces règles de sélection se déduisent directement des contraintes et des relations économiques entre domaines.

Dans un contexte où tous les domaines peuvent avoir des *peers*, une règle suffisante est de toujours préférer les routes passant par un client aux routes passant par un *peer* ou un fournisseur. Dans le cas plus courant où les petits domaines ne peuvent pas avoir de relation de *peering* avec des domaines bien plus grands (des Tier-3 avec des Tier-1 par exemple), la précédente condition peut-être relaxée et les domaines peuvent avoir la même préférence pour les routes passant par des *peers* que pour les routes passant par les clients. Néanmoins, les routes passant par des domaines fournisseurs doivent toujours avoir la préférence la plus faible. Ces conditions sont en accord avec les intérêt économiques d'un domaine,

celui-ci préfèrera passer par une route sans payer (clients et *peers*) plutôt que de payer le transit (en passant par un fournisseur).

Gao a également donné des conditions de propagation qui garantissent la convergence et la stabilité de BGP [47]. Une condition suffisante de convergence est de ne propager que les routes *valley-free* (sans vallées). Une route est *valley-free* si elle se compose de trois parties : une première partie (éventuellement vide) de liens client-fournisseur, suivie d'un seul lien de *peering* au maximum, suivi d'une partie (éventuellement vide) de liens fournisseur-client. Ces routes ne doivent contenir aucun lien fournisseur-client qui précède un lien client-fournisseur. La figure 1.5(a) montre des routes *valley-free* (en vert) dont la propagation permet la stabilité de BGP. La figure 1.5(b) montre des routes contenant des « vallées » et qui ne doivent pas être propagées sous peine d'une instabilité de BGP.



(a) Routes correctes *valley-free*.

(b) Routes incorrectes non *valley-free*.

FIGURE 1.5 – Exemples de routes dans un réseau inter-domaine hiérarchique.

1.4.4.4 Scalabilité de BGP

La scalabilité de BGP dépend de deux paramètres principaux : la taille de la table de routage et le nombre de messages échangés par les routeurs.

Taille de la table de routage : Bien que la taille des tables de routage dépende du nombre de domaines, elle contient actuellement 10 fois plus de préfixes qu'il n'y a de domaines. La raison en est le *multihoming*, le fait que beaucoup de domaines disposent de plusieurs préfixes avec différentes routes pour les joindre. Le but du *multihoming* est de diversifier les routes et de disposer de routes alternatives en cas de panne ou de congestion. Néanmoins, cela augmente considérablement la taille des tables de routages. La figure 1.6 montre l'évolution de la taille des tables de routage de 1994 (année de l'introduction du CIDR [10], qui permet de regrouper les adresses en préfixes [46]) jusqu'au début de l'année 2014. L'évolution était quasi-linéaire jusqu'en 1998 puis est devenue sur-linéaire jusqu'en 2001. A cette date, plusieurs domaines ont pris des mesures à court terme pour ralentir la croissance de la taille des tables de routage [27]. La courbe semble croitre linéairement ces dernières années, mais son évolution à long terme est encore mal comprise.

Nombre de messages échangés : Les messages échangés par les routeurs BGP sont principalement des mises à jour. Le nombre de ces messages dépend donc des changements intervenant dans le réseau : mise à jour de la topologie (nouveau lien, suppression d'un lien, etc.), panne de nœuds ou de liens, changement de politique de routage, etc.

1.4.5 L'écosystème d'Internet

Avant de discuter de la mise en place de la QoS dans un réseau inter-domaine comme Internet, il faut décrire le modèle économique actuel et son influence sur l'évolution du réseau.

1.4.5.1 Les acteurs économiques

Comme nous l'avons vu précédemment, les domaines sur Internet peuvent être classés en trois catégories :

10. CIDR : Classless Inter-Domain Routing.

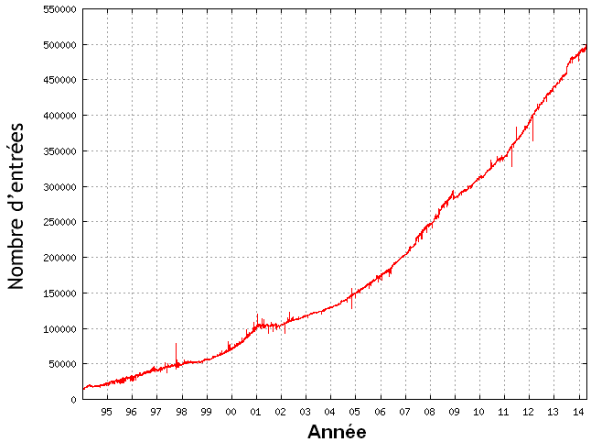


FIGURE 1.6 – Evolution de la taille des table de routage BGP (en nombre de préfixes) de 1994 à 2014 [5].

i Les domaines Tier-1 qui fournissent une connectivité à tout le réseau et qui servent principalement de domaines de transit aux autres,

ii Les domaines Tier-2 qui sont clients des domaines Tier-1. Ce sont pour la plupart des opérateurs et fournisseurs d'accès nationaux,

iii Les domaines Tier-3 qui sont pour la plupart des fournisseurs d'accès régionaux, clients de domaines Tier-1 ou Tier-2 et qui paient ceux-ci pour le transit des données à la connectivité.

A ces domaines, il faut ajouter deux acteurs pour que la système économique en inter-domaine soit complet :

- **Les fournisseurs de services ou de contenus :** Ce sont les acteurs qui génèrent une grande partie du trafic sur Internet. Initialement, ils étaient connectés aux domaines Tier-1 mais ils ont tendance à se connecter de plus en plus aux domaines Tier-2 et Tier-3 (nous y reviendrons dans la section suivante). Il en existe plusieurs types qu'on peut regrouper en deux catégories : les fournisseurs de contenus (*Content Providers*,- CP) comme la vidéo (Youtube, Dailymotion, etc.), la musique (Deezer, iTunes, etc.), les encyclopédies en ligne, etc. L'autre catégorie étant les fournisseurs de services comme la géolocalisation (Google Maps), les moteurs de recherche (Google, Bing, etc.), les sites de vente en ligne (Amazon, eBay, etc.) , les réseaux sociaux (Facebook, Twitter, etc.) et tout autre service à valeur ajoutée.
- **Les utilisateurs finaux :** Ce sont les individus ou les entreprises qui consomment les contenus et services fournis par les acteurs sus-cités.

Ces deux acteurs complètent ce qu'on appelle la *chaîne de valeur économique* d'Internet [18]. Les revenus sont générés en amont de cette chaîne au niveau des fournisseurs d'accès et de services : soit par la publicité en ligne (comme le moteur de recherche Google), soit par le paiement des utilisateurs finaux (comme Amazon).

1.4.5.2 Retour sur la structure hiérarchique d'Internet

La topologie d'Internet est en train de changer. Bien que, techniquement, le modèle hiérarchique vu en figure 1.2 est encore valide, en ajoutant les fournisseurs de services et de contenus, le schéma change. La figure 1.7 montre les nouvelles interconnexions constatées ces dernières années. En plus des interconnexions classiques entre domaines (ici en gris), on peut constater deux changements importants :

i Les fournisseurs de services et de contenus, qui au départ étaient connectés au domaines Tier-1 pour pouvoir atteindre tous les domaines d'Internet, mettent de plus en plus souvent en place des liens de *peering* avec les domaines Tier-2 et Tier-3 [68] (en connectant des caches et des CDN [11]). Ceci afin d'être plus proches des utilisateurs finaux. Cela permet une meilleure QoS (en ne traversant qu'un seul domaine, il est plus facile de maintenir une QoS suffisante), mais cela permet également de court-circuiter les domaines de transit (Tier-2 et surtout Tier-1) et ainsi ne pas payer le prix du transit des données.

11. CDN : Content Distribution Networks

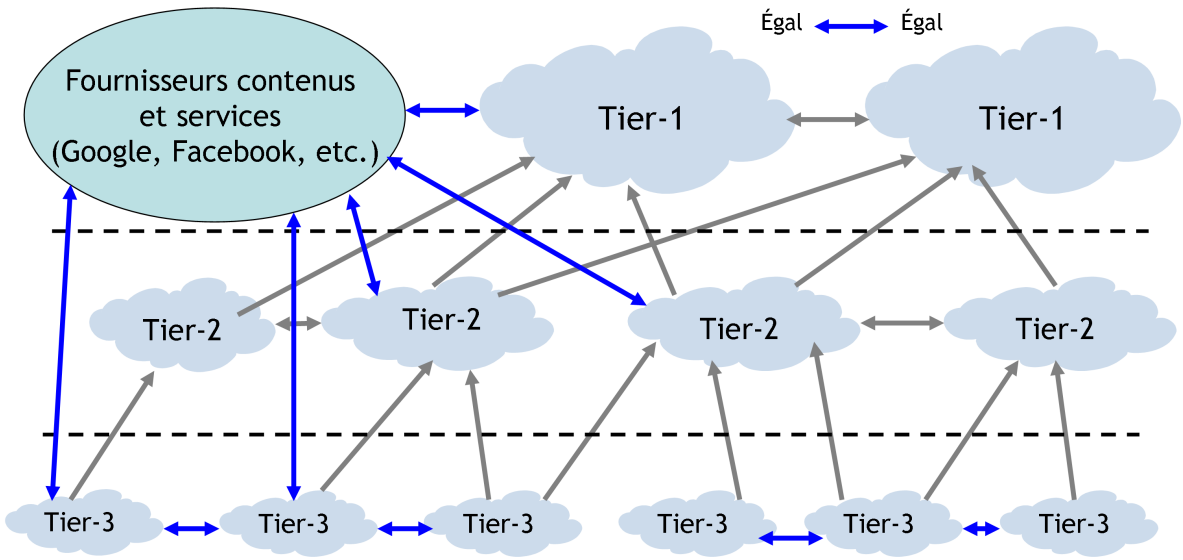


FIGURE 1.7 – Nouvelle structure d'Internet.

ii Les domaines Tier-3 voisins développent de plus en plus souvent des liens de *peering* entre eux selon une règle simple : si l'établissement et l'entretien du lien de *peering* coûtent moins cher que le transit des données, alors il faut préférer le lien de *peering*. Ce qui fait que pour une communication ou un transfert de données dans la même région géographique, les domaines de transit sont court-circuités.

On peut voir ces nouveaux liens (en bleu) sur la figure 1.7. On peut voir qu'Internet est de moins en moins hiérarchique est qu'il ressemble de plus en plus à un réseau maillé : on dit qu'Internet devient *plat* (*flat Internet*) [38].

1.4.5.3 Partage des revenus et profit des domaines de transit

Comme on l'a vu dans la section précédente, les domaines Tier-3 ont tendance à établir des liens de *peering* entre eux. De plus, les fournisseurs de services et de contenus se connectent de plus en plus directement aux fournisseurs d'accès Internet et court-circuitent également les grands domaines de transit. Cette situation entraine un manque à gagner pour les domaines de transit.

Mais ce n'est pas la seule difficulté, car il se trouve également que le prix du transit s'effondre littéralement depuis plus de dix ans. Le prix de transit, qui se mesure en dollars par Mbps, est passé de $1200 par Mbps en 1998 à $0,94 en 2014 [88]. La figure 1.8 montre l'évolution du prix de transit durant cette période. On peut constater que ce prix baisse d'environ 30% par an. Même s'il faut pondérer cette baisse par l'augmentation exponentielle du trafic sur Internet (ce qui fait que le prix multiplié par la quantité de données ne baisse pas aussi drastiquement), les revenus des domaines de transit sont en baisse permanente [88].

L'explosion du trafic sur Internet a un autre effet, négatif cette fois, sur les revenus et les bénéfices des domaines de transit et des opérateurs. En effet, pour pouvoir supporter un trafic croissant et des services et contenus demandant une certaine QoS (vidéo, jeux en ligne, etc.), les domaines de transit et les opérateurs sont obligés de surdimensionner leurs réseaux, c'est-à-dire le concevoir avec une capacité beaucoup plus élevée que le trafic actuel, afin de pouvoir offrir une qualité d'expérience acceptable aux utilisateurs finaux. Bien qu'il n'y ait actuellement aucune garantie de QoS dans les réseaux inter-domaine (et en particulier Internet), le surdimensionnement permet de satisfaire les utilisateurs finaux et faire fonctionner correctement les applications et services nécessitant de la QoS. Ce surdimensionnement a néanmoins un coût élevé et qui est totalement assumé par les domaines de transit et les opérateurs.

Les domaines investissant financièrement dans le surdimensionnement devraient avoir un retour sur investissement pour les inciter sur cette voie. Or, il n'en est rien : en 2008 déjà, plus de 60% des revenus générés par Internet étaient directement récupérés par les fournisseurs de services et de contenus [18]. Cette situation crée un déséquilibre économique et n'incite pas les domaines de transit et les opérateurs à investir davantage afin de pérenniser le bon fonctionnement des réseaux.

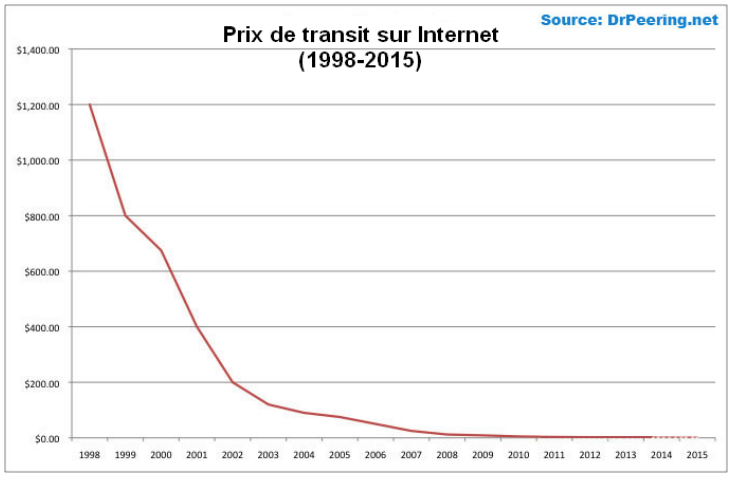


FIGURE 1.8 – Evolution du prix du transit en $ de 1998 à 2014 et projection pour 2015 [88].

Les domaines de transit et les opérateurs obtenaient un retour sur investissement tant que le nombre de réseaux clients (pour les domaines de transit) et d'utilisateurs finaux (pour les opérateurs et fournisseurs d'accès) augmentait de manière importante. Or, le taux de pénétration d'Internet ainsi que d'autres services de télécommunication devient important et le nombre d'utilisateurs et de clients stagne dans certains domaines, particulièrement dans les pays industrialisés.

Tous ces facteurs créent des tensions et des conflits économiques entre fournisseurs de services et de contenus et domaines de transit/opérateurs, mais également entre les domaines de transit eux-mêmes. En effet, sur les routes comportant plusieurs domaines de transit, le conflit entre le fournisseur de services et de contenus et le premier domaine de transit peut se déplacer en aval et entrainer un conflit entre deux domaines de transit. Ce fut le cas notamment dans l'affaire opposant le domaine de transit Cogent à Orange en 2011 et 2012 [4]. MegaUpload [12] produisait un très grand volume de données qui transitait par Cogent (avec lequel MegaUpload avait un contrat), puis, via un lien de *peering*, par Orange, soit à destination de ses utilisateurs finaux, soit à destination d'autres domaines pour lesquels Orange assurait le transit. Cogent a voulu redimensionner le lien de *peering* avec Orange afin de maintenir un débit suffisant pour le téléchargement depuis MegaUpload. Orange a refusé, arguant que le trafic était totalement déséquilibré et que Cogent envoyait treize fois plus de trafic qu'il n'en recevait d'Orange. Le refus d'Orange était motivé par le fait que le redimensionnement du lien de *peering* (et, au-delà, le redimensionnement de tout le réseau pour supporter le surplus de trafic) était un investissement qui ne rapporterait pas de revenus, car le maintien du service de MegaUpload ne bénéficiait qu'à celui-ci. L'affaire fut portée en Justice et Orange a eu gain de cause. Il y a eu depuis plusieurs autres conflits de ce genre, appelés *peering disputes*, entre domaines.

Cette affaire traduit le déséquilibre actuel dans le système économique d'Internet. Les fournisseurs de services et de contenus récupèrent la plus grande partie des revenus générés par Internet, tandis que les investissements lourds pour le surdimensionnement sont assumés par les domaines de transit et les opérateurs, dont les revenus sont de plus en plus limités.

1.4.6 La QoS en inter-domaine

1.4.6.1 La QoS en tant que valeur ajoutée

Une manière de générer de nouveaux revenus pour les domaines de transit et les opérateurs est de créer un service générant de la valeur ajoutée. Les services avec garantie de QoS peuvent y contribuer. En effet, bien que la qualité d'expérience des utilisateurs soit correcte grâce au surdimensionnement, il n'y a aucune garantie de QoS. Proposer comme un service des routes avec QoS garantie permettrait de créer une nouvelle valeur ajoutée dans le réseau et ainsi disposer d'une nouvelle source de revenus tout en augmentant la qualité d'expérience des utilisateurs.

12. Site de téléchargement en ligne, disparu depuis.

L'existence d'un marché pour la QoS en inter-domaine est évidente. Plusieurs études ont montré qu'une grande partie des utilisateurs finaux en Europe est prête à payer davantage pour une QoS assurée [103, 102]. D'autres études ont montré qu'une tarification différenciée et basée sur la qualité du prix de transit entre domaines améliorerait non seulement les revenus des domaines de transit, mais également l'utilisation des ressources et l'expérience des domaines clients [112]. Par ailleurs, certains utilisateurs payent déjà certains services qui nécessitent des garanties de QoS (VoD, jeux en ligne, etc.).

L'introduction de la QoS en inter-domaine n'a évidemment pas pour but unique de créer une nouvelle source de revenus. Les avantages sont multiples, parmi lesquels [11] :

i **Optimiser l'utilisation des ressources réseau :** Le surdimensionnement actuel des réseaux fait que, la plupart du temps, ces ressources sont sous-utilisées. Une nouvelle architecture assurant la QoS permettrait par la même occasion de mieux gérer les ressources car les besoins seraient connus et contractualisés.

ii **Permettre l'émergence de nouveaux types de services :** Bien que certains services nécessitant une QoS existent et fonctionnent correctement pour l'instant, la mise en place d'une QoS garantie permettrait l'émergence de nouveaux services pour lesquels cette garantie est indispensable comme la médecine à distance pour laquelle la QoS et la sécurité sont critiques. Elle permettrait également d'améliorer la fiabilité des services déjà existants.

iii **Maintenir l'incitation à l'investissement dans les réseaux :** Si le déploiement de la QoS en inter-domaine assure un retour sur investissement, cela aura pour effet d'encourager les opérateurs et les domaines dans cette voie, et ainsi de permettre une évolution du réseau vers une plus grande performance et une plus grande fiabilité.

1.4.6.2 Difficultés de déploiement de la QoS en inter-domaine

Les limites du surdimensionnement : Se contenter du surdimensionnement comme c'est le cas jusqu'à présent est impossible. Nous avons déjà évoqué les problèmes économiques que le surdimensionnement provoque et le fait que les domaines n'ont pas d'incitation financière à continuer sur cette voie. Mais il y a également des problèmes techniques. Il y a une limite théorique de capacité de transmission connue sous le nom de *limite de Shannon* [104]. Ce qui fait que l'augmentation de la capacité de transmission (bande passante) ne pourra pas augmenter de façon continuelle. On observe déjà que l'augmentation annuelle de la capacité des équipements optiques ralentit [116]. Assurer la QoS doit se faire par une meilleure gestion des ressources car le surdimensionnement n'est pas pérenne aussi bien sur le plan économique que technique.

Adaptation de BGP à la QoS : Plusieurs travaux ont été proposés pour étendre BGP au support de la QoS [24, 118, 25]. Néanmoins, deux principaux problèmes se posent à ces extensions :

- **La scalabilité :** Comme expliqué en section 1.4.4.4, la scalabilité est un aspect critique pour BGP. Il se trouve que BGP ne permet qu'une seule route pour une destination donnée, et rien ne garantit que cette route dispose des caractéristiques de QoS souhaitées. La solution communément proposée est de définir plusieurs classes de QoS et de modifier BGP afin qu'il calcule une route vers chaque destination pour chaque classe de QoS. L'inconvénient est que la taille de la table de routage est multipliée par le nombre de classes de QoS (qui, vu les différentes combinaisons possibles de paramètres, peut-être très élevé). Le nombre de messages échangés entre routeurs serait aussi multiplié par le nombre de classes. Cette solution n'est pas réalisable à l'échelle d'Internet.
- **La dynamicité et l'instabilité :** L'autre problème concernant l'adaptation de BGP à la QoS est la dynamicité des paramètres de QoS. Non seulement les besoins et contraintes des applications réseaux peuvent varier, nécessitant la définition de nouvelles classes de QoS ou un passage rapide d'une classe à l'autre, mais aussi les caractéristiques des liens et des routes varient en fonction du trafic et de la congestion. Par exemple, un route étant classée avec une bande passante et un délai donnés peut voir sa bande passante se réduire considérablement et son délai augmenter à cause du trafic qui y passe, il faudrait donc repropager cette route avec les nouveaux paramètres de QoS, qui ne seront pas non plus constants vu que le trafic est variable. Outre le très grand nombre de messages de mises à jour que ça engendrerait, il se pourrait que les changements de paramètres soient si fréquents que BGP n'ait pas le temps de converger entre deux changements, ce qui rendrait le routage instable.

***DiffServ* en inter-domaine :** Comme nous l'avons vu en section 1.3.2, DiffServ permet de déployer une certaine QoS en intra-domaine. Le passage à l'inter-domaine est compliqué, et comme toutes les autres solutions que l'on veut généraliser à l'inter-domaine, le problème principal est le même : **la coordination**. En effet, pour que *DiffServ* soit déployé à l'échelle inter-domaine, il faudrait que la définition des classes soit la même dans tous les domaines, c'est-à-dire que tous les domaines utilisent le même code pour une classe donnée. Il faudrait surtout que le PHB et la politique de traitement par paquets soient uniformisés. En effet, pour un paquet appartenant à une classe donnée, un domaine peut très bien définir une politique de priorité absolue (pour des raisons économiques ou autres), alors que le domaine suivant sur la route considère cette classe comme non prioritaire. Les intérêts des domaines étant divergents (principalement pour des raisons économiques), rien ne les incite à accorder le même traitement et la même priorité aux classes de paquets.

1.4.7 Possible solution : la composition de SLA

Une solution possible pour pallier le problème de coordination est de séparer les problématiques de QoS en intra et inter-domaine. Chaque domaine peut définir un ensemble de routes entre ses voisins avec une certaine garantie de QoS. Ces routes étant internes (elles ne traversent que le domaine en question pour relier deux de ses voisins), le domaine en question peut assurer la QoS avec les mécanismes habituels (*DiffServ* ou autre). Au niveau inter-domaine, une route globale peut être obtenue en mettant bout à bout (ou en composant) les routes internes aux domaines traversés de façon à satisfaire des contraintes de QoS globales. La mise en place des routes internes étant rémunérée, les domaines ont une incitation à respecter la QoS promise.

1.4.7.1 Définition d'un SLA

Les routes internes évoquées dans le paragraphe précédent sont formalisées par la notion de *Service Level Agreement* (SLA). Un SLA est un contrat proposé par un domaine fournisseur à un domaine voisin appelé *client*. Ce contrat assure au client que le fournisseur acheminera son trafic avec une QoS spécifiée, pendant une certaine durée jusqu'à un point d'interconnexion spécifié, et ceci contre une rétribution financière.

Les SLA contiennent deux parties :

1. **Caractéristiques techniques :** Cette partie définit la QoS à respecter par le SLA (délai, bande passante, fréquence de pannes, etc.) les points d'interconnexion, etc.
2. **Caractéristiques économiques et juridiques :** Dans cette partie, on trouve le prix du SLA, les éventuelles indemnités en cas de non respect de la QoS, la durée du SLA, etc.

1.4.7.2 *Monitoring* et violation de SLA

Le *monitoring*[13] d'un SLA consiste à surveiller le trafic acheminé dans le cadre de ce SLA et vérifier que la QoS et toutes les caractéristiques techniques sont respectées. Si on constate que la bande passante est en dessous du paramètre spécifié dans le SLA ou que le délai est plus long que le délai maximum spécifié, on dit qu'il y a *échec* ou *violation* du SLA. Dans ce cas, le domaine client est remboursé et/ou le domaine fournisseur lui verse des indemnités selon ce qui est spécifié par le SLA. Certains travaux [96] proposent de définir des classes de violation partielle (violation d'un paramètre pendant une courte période, par exemple) et totales (échec de l'acheminement des données) et de définir des politiques de pénalités en conséquence.

1.4.7.3 Composition de SLA

Pour établir une route inter-domaine avec garantie de QoS, il faut construire une *chaîne* de SLA, c'est-à-dire mettre bout à bout les SLA de différents domaines pour former une route du domaine client jusqu'au domaine destinataire. C'est ce qu'on appelle la composition de SLA.

La figure 1.9 montre un réseau inter-domaine traversé par une route composée de SLA. Chaque domaine de transit dispose d'un ensemble de SLA (routes en pointillé traversant le domaine). Le SLA en rouge est celui qui a été choisi pour construire la chaine. Ainsi, les SLA 1, 6 et 10 composent la route avec

13. Surveillance.

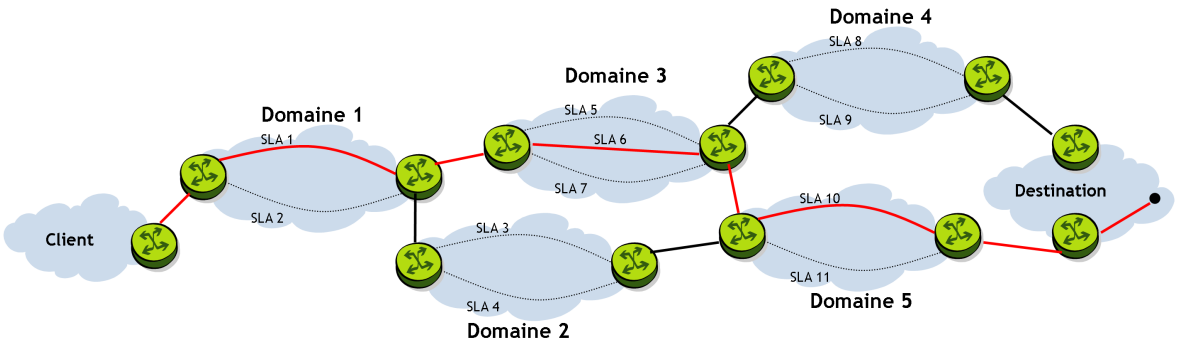


FIGURE 1.9 – Composition d'une chaîne de SLA et établissement d'une route inter-domaine avec QoS.

QoS garantie qui relie le domaine client (qui peut être lui-même un domaine de transit ou un fournisseur de contenus et de services) au domaine destination.

Avant de se demander quels SLA choisir pour construire la route, il faut définir les paramètres que cette route doit respecter en matière de QoS. Il s'agit d'un problème d'optimisation. Numérotons les SLA de 1 à n et désignons par b_i, d_i et ℓ_i respectivement la bande passante, le délai et le taux de perte associé au sla i, et soit p_i sont prix. Ces paramètres sont choisis à titre d'exemples pour les raisons évoquées en section 1.2.1. Supposons que le client ait besoin d'une route dont la bande passante est supérieure à $b_{\min}$, le délai inférieur à $d_{\max}$ et le taux de perte inférieur à ℓ_{max}. Le client voudra minimiser le prix à payer tout en garantissant la QoS demandée. Le problème d'optimisation est donc le suivant :

$$\min \sum_{\text{SLA } i \in \text{ route}} p_i$$
$$t.q. \begin{cases} \min_{\text{SLA } i \in \text{ route}} b_i \geq b_{\min} \\ \sum_{\text{SLA } i \in \text{ route}} d_i \leq d_{\max} \\ \prod_{\text{SLA } i \in \text{ route}} (1 - \ell_i) \leq 1 - \ell_{\max} \end{cases} \tag{1.1}$$

Le problème de décision associé au problème 1.1 est NP-complet. Plus généralement, trouver un chemin satisfaisant une certaine QoS dans un graphe où les arêtes sont associées à des paramètres de QoS est NP-complet s'il y a au moins deux contraintes additives et/ou multiplicatives [50, 113]. Hormis la recherche exhaustive pour trouver la meilleure solution, deux approches sont souvent citées :

- **Heuristiques et métaheuristiques :** Il s'agit de résoudre le problème de façon centralisée (en supposant que les domaines publient leurs SLA) en trouvant des solutions acceptables ou proches de l'optimum.
- **Algorithmes distribués :** Cette approche convient mieux aux exigences de confidentialité des opérateurs. Elle consiste à créer la chaîne de SLA à partir de données locales uniquement, et d'aboutir à la route de bout en bout. Aucun domaine en particulier n'aura connaissance de la totalité de la topologie et des ressources du réseau inter-domaine. Un exemple d'approche distribuée est l'approche en cascade : chaque domaine reçoit un bout de route menant jusqu'à lui, il choisit un de ses SLA et l'ajoute à la route partielle, puis c'est au domaine suivant de faire de même jusqu'à ce que le domaine destination soit atteint.

Ces approches seront détaillées dans la section suivante, dans la présentation du projet européen dans le cadre duquel s'est déroulée cette thèse.

1.5 Le projet ETICS

Le projet européen ETICS[70, 8] a été créé pour proposer des solutions économiques et techniques combinées afin de déployer la QoS en inter-domaine. Cette thèse s'est déroulée dans le cadre de ce projet.

Dans un réseau inter-domaine, les aspects économiques influencent grandement les aspects techniques (par exemple le routage), qui à leur tour ont une influence sur les développements techniques. Une solution

permettant le déploiement de la QoS en inter-domaine doit traiter les deux aspects de manière cohérente et complémentaire. Plus précisément, les objectifs du projet ETICS sont les suivants :

- Proposer de nouveaux modèles économiques basés sur le marché de la QoS dans les réseaux inter-domaine,
- Concevoir des solutions techniques (architecture, algorithmes de routage, etc.) permettant d'obtenir des routes avec garantie de QoS en inter-domaine, ceci en prenant en compte l'hétérogénéité des technologies et protocoles utilisés par chaque domaine,
- Démontrer la faisabilité et l'efficacité de ces solutions sur un banc d'essai à large échelle.

Un tel système, s'il est déployé, bénéficierait à tous les acteurs économiques :

- Les opérateurs et domaines de transit proposeront la QoS comme nouvelle valeur ajoutée, ce qui leur fournira une nouvelle source de revenus et les incitera à investir davantage dans le développement du réseau,
- Les fournisseurs pourront proposer des services et contenus avec garantie de QoS, ce qui augmentera leur fiabilité et permettra la création de nouveaux services jusqu'ici impossibles à cause de l'absence de QoS, ce qui est économiquement rentable pour ces fournisseurs,
- Les utilisateurs finaux auront accès à plus de services et de meilleure qualité. Ils pourront également choisir la qualité qui leur convient le mieux si plusieurs classes de QoS sont définies.

1.5.1 Les modèles économiques proposés dans ETICS

Le projet ETICS préconise le principe du *Sending Party Network Pays* (SPNP) aux points d'interconnexion entre domaines pour l'échange de trafic avec QoS (le trafic *best effort* peut continuer à être géré de la manière actuelle). Cela signifie que celui qui émet le trafic doit payer celui qui le reçoit et l'achemine jusqu'au prochain point d'interconnexion tout en assurant une QoS donnée. Cela signifie que c'est le fournisseur de contenus qui paiera le trafic jusqu'au domaine destinataire.

Trois types de relations sont définis entre les domaines sur lesquels cette solution pourrait être déployée : *association ouverte*, *fédération* et *alliance*. Les informations échangées ainsi que le partage des revenus dépend du type de relations mis en place :

1. **Association ouverte :** Si les domaines forment une association ouverte, les informations échangées entre eux se limitent au strict minimum. Ainsi, chaque domaine échangera des informations de connectivité et de QoS avec ses voisins uniquement. La coordination nécessaire doit être minimale, comme par exemple l'unification des politiques *DiffServ*. Ce type de relation est plus approprié au début de la mise en place du système, car il nécessite le minimum de confiance entre les domaines.
2. **Fédération :** Ce type de relation permet aux domaines d'échanger des informations sur le réseau au-delà de leurs seuls voisins. La propagation d'informations dans toute la fédération permet aux domaines d'avoir une vue d'ensemble du réseau. Un des domaines peut alors se charger du calcul de la route et de la composition de SLA.
3. **Alliance :** Dans une alliance, les informations de connectivité, de QoS ainsi que des paramètres économiques sont dévoilés à l'ensemble des domaines membres. Ces informations peuvent être centralisées au sein d'une tierce partie qui composera les chaînes de SLA et négociera avec les clients. Ce type de relation nécessite un haut niveau de confiance entre les domaines et ne peut être mis en place que dans un marché de QoS déjà mature.

Si la composition de SLA se fait de manière distribuée, chaque domaine fixera lui-même la part des revenus de façon à maximiser son gain, mais également à minimiser les risques d'échec de la composition (en cas de désaccord entre les domaines ou de refus du client). Si la composition de SLA est centralisée, un mécanisme de redistribution des revenus doit être mis en place de façon à satisfaire le maximum de domaines, comme un partage basé sur la valeur de Shapley [106] ou d'autres mécanismes [17].

Les scénarios centralisés, et particulièrement la désignation d'une entité centralisée neutre, ne sont évidemment pas envisageable à l'échelle de l'Internet mondial. Il est plus réaliste d'envisager une coexistence de différents scénarios ainsi que de plusieurs communautés (associations, fédérations et/ou alliances) pour couvrir l'ensemble d'Internet.

1.5.2 Les solutions techniques proposées dans ETICS

La première question qui se pose est celle de la définition et du calcul des SLA : doit-on les calculer a priori, c'est-à-dire avant de recevoir une demande de route avec QoS, ou bien doit-on les calculer après

une telle demande ? Les deux scénarios sont envisageables et ont été étudiés dans le projet ETICS :

- **Calcul a priori ou scénario *push* :** Dans ce cas, les domaines identifient leurs connexions, leurs routes internes et leurs ressources en QoS et établissent un *catalogue* de SLA prêts à l'emploi. Ce catalogue peut-être propagé ou non selon des scénarios que nous détaillerons plus loin. Lorsqu'un client fait une requête de route avec QoS, ces SLA pré-calculés sont utilisés pour calculer la route. Ce scénario nécessite une bonne connaissance des ressources internes du domaine et des types de requêtes (profils de clients). Il est plus adapté à un marché mature où le domaine peut s'appuyer sur un historique de demandes et d'offres pour définir au mieux ses SLA.
- **Calcul a posteriori ou scénario *pull* :** Dans ce cas, les SLA possibles ne sont établis que lorsqu'un domaine reçoit une requête de route avec QoS. Il doit identifier les routes internes et ressources dont il dispose pour faire une offre au client. Ce scénario est adapté à une situation où le domaine ne connait pas bien les profils des clients et n'a pas d'historique sur lequel se baser.

Ces deux scénarios peuvent se conjuguer avec des solutions centralisées ou distribuées pour la composition de SLA et le calcul des routes. Ils s'appliquent différemment selon que les domaines forment une association, une fédération ou une alliance. Les quatre solutions suivantes sont proposées :

1.5.2.1 Scénario *pull* avec calcul distribué

Cette solution implique qu'à la réception d'une requête par un client, le domaine décide des ressources à allouer à la requête et du budget qu'il s'attribue puis envoie ces informations à un de ses voisins qui dispose d'une route vers la destination. Le domaine voisin décide des ressources à attribuer et du budget à son tour, puis agrège ces informations avec celles reçues par le précédent domaine et envoie le tout à un domaine voisin. Le calcul et l'agrégation se font ainsi de suite : chaque domaine reçoit des informations du précédent domaine, choisit ses ressources, agrège ces informations avec les précédentes et envoie le tout au domaine suivant jusqu'à ce que la destination soit atteinte. C'est un processus en **cascade** qui peut aussi se faire dans le sens inverse en partant de la destination jusqu'à atteindre le client.

L'intérêt de cette solution est qu'elle nécessite très peu de divulgations d'informations de la part des domaines. Les domaines reçoivent des informations agrégées et ne savent donc pas quelles sont les ressources accordées par tel ou tel domaine en particulier.

Cette solution pose néanmoins un problème de *monitoring*. En cas d'échec de la composition de la chaîne de SLA, il est difficile de savoir quel domaine en est responsable (en accordant trop peu de ressources ou en prenant une trop grande part du budget). Des mécanismes supplémentaires doivent être mis en place pour permettre un *monitoring* efficace. De plus, la position dominante du premier domaine sur la chaîne pose des problèmes d'équité car il risque de prélever le maximum de budget et fournir le minimum de ressources.

Cette solution est à privilégier en cas d'association ouverte formée par les domaines, car elle nécessite peu d'échange d'information et ne s'appuie pas sur un historique de requêtes.

1.5.2.2 Scénario *push* avec calcul distribué

Dans cette solution, les SLA sont pré-calculés par les domaines, puis publiés aux domaines voisins. Les domaines recevant des SLA les agrègent aux leurs et propagent ces SLA agrégés à leur tour. La propagation se fera jusqu'à ce que chaque domaine ait un catalogue de routes avec QoS pour toutes les destinations. Lorsqu'un domaine reçoit une requête d'un client, il propose une (ou plusieurs) route de son catalogue au client qui choisira. Ici, la composition de SLA se fait par propagation avant la réception d'une requête.

Ce scénario présente le même intérêt de confidentialité que le précédent, les SLA étant agrégés avant d'être propagés, il est difficile d'avoir des informations sur les ressources d'un domaine en particulier. Néanmoins, en plus du problème de *monitoring* qu'elle pose, cette solution peut présenter des problèmes de scalabilité. En effet, elle reprend les principes de l'extension de BGP à la QoS (propagation par classe et par route) et pose le même problème : l'explosion du nombre de routes.

Si le problème de scalabilité ne se pose pas (peu de domaines et/ou peu de SLA par domaine), cette solution conviendrait à une association ouverte de domaines, mais qui serait assez mature pour disposer d'un historique de requêtes et ainsi adapter les SLA et les routes aux profils des clients.

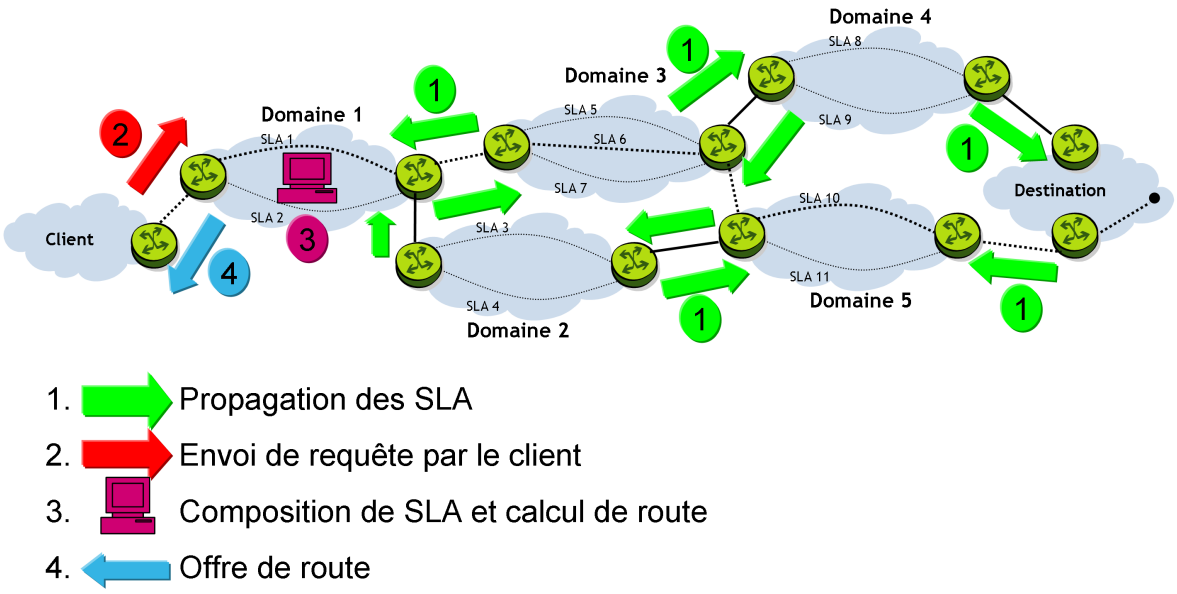


FIGURE 1.10 – Illustration du scénario *push centralisé par domaine*.

1.5.2.3 Scénario *push* centralisé par domaine

Dans cette solution, chaque domaine propage ses SLA pré-calculés jusqu'à ce que tous les autres domaines les connaissent, mais sans agrégation ni composition de SLA. Ainsi, chaque domaine connait les SLA de tous les autres domaines de la communauté (fédération ou alliance). Si le client émet une requête, c'est le domaine recevant la requête (un domaine connecté au client, donc) qui se charge de composer les SLA et de construire la chaîne de SLA de manière centralisée.

La figure 1.10 illustre ce scénario. L'avantage ici est que la composition de SLA se fait de manière centralisée par un domaine qui a une vision globale du réseau. Les inconvénients sont la divulgation des SLA (et donc de certaines ressources) par les domaines, ainsi que la non neutralité du domaine qui effectue la composition de SLA. Il pourrait en effet privilégier ses intérêts personnels (en bridant un concurrent par exemple) plutôt que l'intérêt global des domaines. Cette solution est donc possible uniquement dans le cadre d'une fédération ou d'une alliance dans laquelle les domaines ont des relations de confiance.

1.5.2.4 Scénario *push* totalement centralisé

Ce scénario nécessite la présence d'une entité centralisée neutre qui représenterait les domaines. Les SLA ne sont plus propagés entre les domaines mais publiés à cette entité. Le client envoie sa requête à cette entité qui calculera la route et composera les SLA de manière centralisée. La route obtenue est proposée au client, et s'il l'accepte, les différents domaines impliqués mettront en place la route calculée. Ce scénario est illustré par la figure 1.11.

Cette solution évite le problème que peut poser le calcul de route par un seul domaine. Néanmoins, l'entité neutre doit être agréée par tous les acteurs, cette solution est envisageable dans le cadre d'une alliance.

D'autre scénarios ont été étudiés par ETICS : *pull* totalement centralisé, *pull* centralisé par domaine, etc. Ils n'ont pas été recommandés à cause de considérations économiques (difficultés dans le partage des revenus). Les solutions et architectures détaillées du projet ETICS peuvent être consultées dans le délivrable D4.4 du projet [12].

1.5.3 ETICS et la Net neutralité

Bien que cette thèse ne traite pas directement de la Net neutralité, elle se trouve concernée car les solutions qu'elle propose (et que propose le projet ETICS plus généralement) peuvent être contraintes, voire interdites, par de nouvelles règles de régulation.

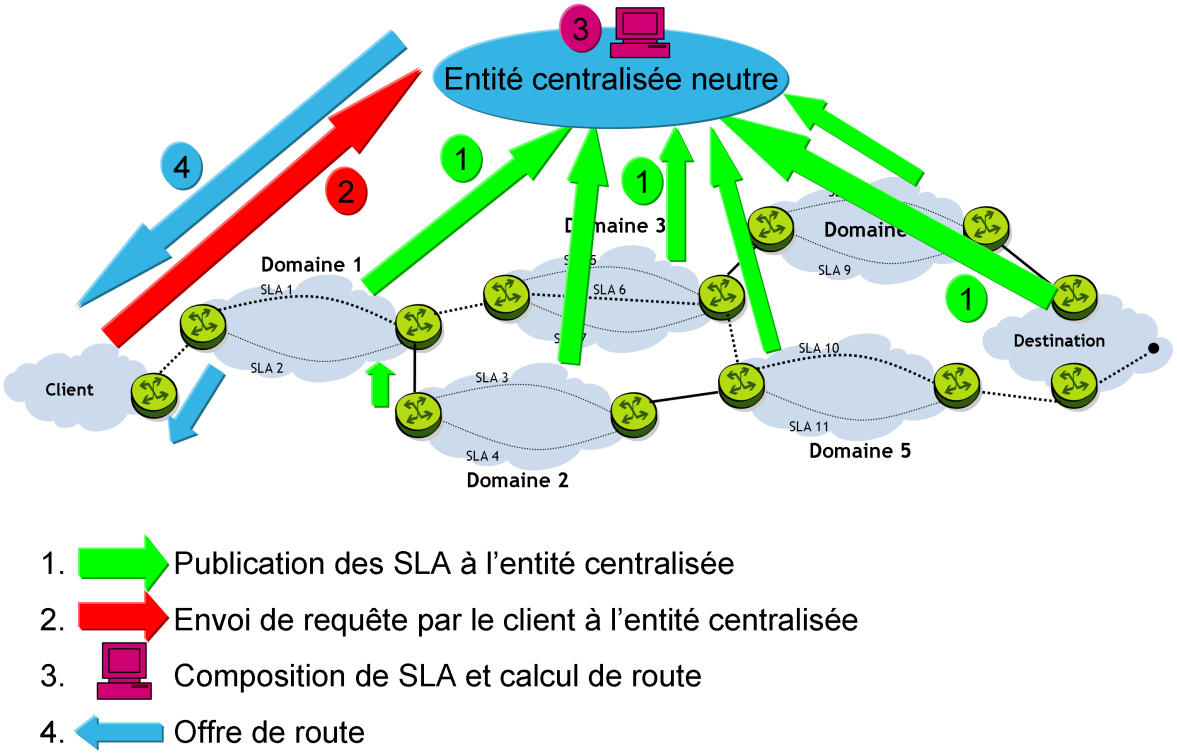


FIGURE 1.11 – Illustration du scénario *push totalement centralisé*.

Pour rappel, la Net neutralité est le principe que le réseau Internet doit traiter exactement de la même manière les données qu'il transporte. Cela exclut une discrimination des données selon leur contenu, leur source, leur destination, etc. Interprété au sens strict, ce principe interdit la priorisation du trafic, et les technologies comme *DiffServ* seraient interdites.

Ce principe impacte les solutions proposées par ETICS. En effet, la mise en place de SLA nécessite forcément une différenciation de traitement pour assurer différents paramètres de QoS.

Le projet de régulation publié en septembre 2013 par la Commission européenne prévoyait d'autoriser la mise en place de routes avec QoS par les opérateurs [7]. Néanmoins, probablement à cause du contexte politique (approche des élections européennes), le Parlement européen a voté, le 3 avril 2014, des amendements garantissant une application assez stricte de la Net neutralité : une différenciation de trafic dans un réseau faisant partie d'Internet ne peut avoir lieu que sur décision de justice, pour protéger la sécurité du réseau ou pour offrir un service qui ne pourrait pas fonctionner avec un traitement normal [2]. Ceci exclut donc tous les services fonctionnant actuellement (VoIP, vidéoconférence, etc.) et pourrait même remettre en cause les offres Triple Play existantes.

Mais le 24 avril 2014, la *Federal Communication Commission* (FFC), le régulateur des télécommunications aux Etats-Unis, a dévoilé une version préliminaire d'une réglementation Internet qui permettrait aux opérateurs de faire payer les fournisseurs de services et de contenus pour garantir l'accès à leurs sites avec des paramètres de QoS optimaux [3]. Clairement, il s'agit d'autoriser une différenciation du trafic pour offrir des services avec une QoS plus ou moins élevée selon le paiement des fournisseurs.

Parallèlement, des accords violant la Net neutralité commencent à se mettre en place entre des fournisseurs de contenus et de services et des fournisseurs d'accès Internet. On peut citer l'exemple du fournisseur de VOD Netflix qui a passé un accord avec l'opérateur de câble Comcast pour l'acheminement des vidéos en meilleure qualité [9]. L'accord prévoit une rétribution financière de Comcast en échange de la QoS accordée au vidéos de Netflix. Avant, Netflix ne payait que la connectivité à un fournisseur de transit, mais la qualité des vidéos n'était pas suffisante pour certains utilisateurs.

La question de la Net neutralité est débattue dans le milieu politique depuis des années et n'est pas encore tranchée. Certains pays peuvent voter des lois pour la garantir tandis que d'autres l'assouplissent. Les points de vue européen et américain sont pour l'instant contradictoires, mais il est prévisible que les décisions politiques évoluent souvent et dans des directions différentes. Il y donc clairement une absence

de consensus et une instabilité de la réglementation.

Dans ce contexte, le projet ETICS n'est pas remis en cause et ses solutions restent valables. La mise en place de ces solutions est cependant dépendante des évolutions de la réglementation :

- Si la différenciation de traitement du trafic est autorisée sur Internet et que des SLA peuvent être instanciés, les solutions du projet peuvent être directement mises en place sur Internet, sur les mêmes réseaux assurant le transfert de données aujourd'hui,
- Si la réglementation impose une vision stricte de la Net neutralité (du moins assez stricte pour que la mise en place de SLA soit impossible), ces solutions pourront être déployées en parallèle à Internet. Les services, la communication et l'acheminement des données avec QoS se fera sur des réseaux indépendants d'Internet et non soumis à la neutralité. Ainsi, certaines entreprises ou certains opérateurs pourraient former des alliances ou des fédérations qui proposeraient une connectivité et des services avec de la QoS.

1.6 Positionnement de la thèse et contributions

Cette thèse s'est déroulée au sein du projet ETICS et a traité deux problématiques : la négociation de SLA et le calcul de chemins dans un réseau hétérogène (comprenant plusieurs protocoles). Ces problématiques sont détaillées dans cette section.

1.6.1 Négociation de SLA

1.6.1.1 Processus de négociation

Dans les scénarios *push*, il y a une (ou plusieurs) étape de négociation de SLA :

- Dans un scénario totalement centralisé, cette étape intervient quand les domaines proposent leurs SLA à l'entité neutre. Selon les scénarios, il se peut que chaque domaine ne puisse proposer qu'un seul SLA pour faciliter la composition et le calcul de route de l'entité neutre. La question qui se pose alors au domaines est : *quel SLA proposer pour maximiser mes revenus ?* Il serait tentant de ne proposer que le SLA le plus rentable, mais cela peut faire échouer le calcul de la route si ce SLA n'est pas adapté à la demande de QoS ou si le prix est trop élevé,
- Dans un scénario centralisé par domaine, le domaine qui reçoit la requête du client peut disposer de plusieurs routes (composées de SLA agrégés, qu'on peut considérer comme des « super SLA »). Il doit donc choisir quelle route proposer au client,
- Dans un scénario distribué (cascade), à chaque étape du processus de calcul de route, un domaine reçoit une portion de route d'un de ses prédécesseurs et doit décider quel SLA agréger à cette portion. Ici, à chaque étape, le dernier domaine à avoir agrégé son SLA est considéré comme client et le domaine suivant est considéré fournisseur de transit. Ce dernier devient à son tour client dans l'étape suivante et ainsi de suite jusqu'à atteindre la destination. Dans un calcul en cascade, il y a donc autant de négociations de SLA que de domaines sur la route.

Dans ces trois scénarios, le processus dit de négociation est le même : un client (qui peut être lui-même un domaine de transit, selon les scénarios) envoie une requête à un ou plusieurs domaines fournisseurs, chaque domaine fournisseur choisit une offre dans son catalogue et l'envoie au client, le client choisit l'offre qui lui convient le mieux.

La figure 1.12 illustre le processus de négociation de SLA. Le domaine fournisseur peut être unique ou bien il peut y avoir une compétition entre les domaines pour satisfaire une requête.

1.6.1.2 Choix du SLA

Le revenu d'un domaine fournisseur est le prix du SLA qu'il offre. Néanmoins, les critères de choix du SLA sont influencés par de nombreux paramètres :

- Les SLA peuvent échouer et dans ce cas le domaine doit au minimum rembourser ce prix, et peut-être verser des indemnités au client. De ce fait, le revenu associé à un SLA doit être pondéré par sa probabilité de réussite. Un des critères est de choisir un SLA qui a peu de chance d'échouer.
- Les domaines ont des ressources limitées. Proposer un SLA qui consommerait trop de ressources peut dégrader les performances du réseau et l'empêcher de faire d'autres offres pour d'autres clients.

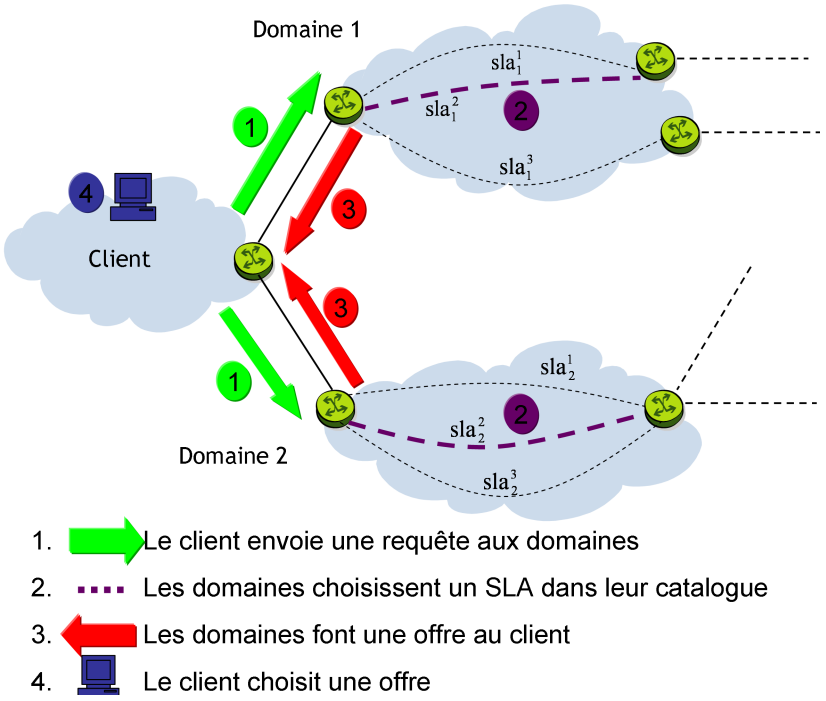


FIGURE 1.12 – Processus de négociation de SLA.

– Il peut y avoir compétition entre certains domaines, il faut donc s'efforcer de proposer le SLA qui a le plus de chance d'être choisi par le client.

Ces critères sont parfois contradictoires, et trouver un juste équilibre pour les satisfaire est difficile car les paramètres sont nombreux.

1.6.1.3 Contributions de la thèse sur la négociation de SLA

Dans la première partie de cette thèse, nous nous plaçons au niveau des domaines fournisseurs et proposons des solutions pour que ces domaines offrent des SLA de façon à maximiser leurs revenus. Dans le chapitre 2, nous étudions d'abord le cas où il n'y a qu'un seul domaine fournisseur pour un client. Sous certaines suppositions, nous proposons une solution optimale afin de maximiser les revenus des domaines à long terme en prenant en compte les critères précédemment évoqués. Puis nous étudions le cas où il y a plusieurs domaines en compétition. Nous donnons une solution optimale dans le cas où ces domaines collaborent ainsi que dans le cas où ils sont égoïstes. Nous montrons que les revenus sont plus élevés dans le cas où ils collaborent, et donc que le Prix de l'Anarchie peut être supérieur à 1, et même arbitrairement grand.

Néanmoins, ces solutions optimales nécessitent une connaissance parfaite du réseau et de beaucoup de paramètres que les domaines ne peuvent pas maîtriser ni connaître : critères de choix du client, état des ressources, ressources des concurrents, etc. Après avoir présenté les principes de bases des algorithmes d'apprentissage, nous adaptons ceux-ci au problème de la négociation de SLA. Grâce à cette solution, les domaines vont *apprendre* quel SLA proposer en fonction de leur expérience passée. L'intérêt de cette méthode est que les domaines n'ont plus à connaître les paramètres du réseau, l'apprentissage est basé sur la seule expérience.

Pour les travaux précédemment cités, notre modèle suppose que le client est sensible au prix du SLA et à la QoS proposée. Il semblait intéressant d'enrichir ce modèle pour qu'il prenne en compte la réputation, car c'est un paramètre crucial dans la vie réelle. Dans le chapitre 3, nous supposons donc que le client est sensible à la réputation du fournisseur et que ceci influence son choix de SLA. La réputation d'un domaine fournisseur est inversement proportionnelle au taux d'échec de ses SLA. Plus les SLA d'un domaine échouent, plus sa réputation baisse, c'est donc une valeur qui évolue avec le temps. Or, la probabilité d'échec d'un SLA dépend des ressources utilisées par le domaine. Plus il vend (ou provisionne) de SLA, plus la probabilité de violation (ou d'échec) est grande. Le problème est donc de trouver un

compromis entre les ressources utilisées et une réputation acceptable. Nous nous proposons d'adapter les algorithmes d'apprentissage vus dans le chapitre précédent de façon à ce qu'ils prennent en compte ce nouveau paramètre. Les simulations montrent que la prise en compte de la réputation influence le choix du client, et donc les stratégies optimales des domaines. Elles montrent également que les algorithmes d'apprentissage sont capables de trouver un compromis entre les ressources utilisées et la réputation afin de maximiser les revenus des domaines. Ces algorithmes se montrent plus performants que des méthodes de l'état de l'art.

1.6.2 Calcul de chemins dans les réseaux hétérogènes et multicouches

1.6.2.1 Hétérogénéité de protocoles dans les réseaux inter-domaine

Une fois la route inter-domaine (en tant que liste de SLA et de domaines) calculée, il faut l'instancier, c'est-à-dire déterminer par quels routeurs et quels liens elle va passer, que ce soit en interne à chaque domaine ou sur les liens d'interconnexion entre les domaines. Si l'utilisation de plusieurs protocoles sur les mêmes équipements est possible, il faut aussi déterminer quels protocoles utiliser sur quelles portions de la route, car il n'est pas rare que les domaines n'utilisent pas les mêmes protocoles, ni les mêmes technologies. Bien que la convergence sur IP permette la communication entre ces réseaux, ignorer d'autres liens d'interconnexion non IP réduit le nombre de solutions possibles. Nous appelons *calcul de chemins* le processus qui permet de déterminer explicitement la route par les équipements et les protocoles utilisés.

Il se trouve que les réseaux sont conçus en couches, avec plusieurs protocoles coexistant sur chaque couche. Or certains protocoles sont incompatibles entre eux et il n'est pas toujours possible de convertir un protocole en un autre. Une des solutions proposées est d'*encapsuler* un protocole dans un autre pour traverser des portions de route qui ne supportent pas le premier protocole, puis de *désencapsuler* le premier du second une fois ces portions traversées. Ces *encapsulations* et *désencapsulations* sont réalisées par des équipements spéciaux et ne sont possibles qu'à certains endroits du réseau. Sur un chemin, il peut y avoir plusieurs encapsulations, dont certaines peuvent être imbriquées.

Calculer un chemin dans ce contexte est complexe car il faut prendre en compte plusieurs contraintes :

- Les encapsulations ne sont possibles qu'à certains endroits du réseau, de même que les désencapsulations,
- Il n'est pas possible d'encapsuler n'importe quel protocole dans n'importe quel autre, seules certaines encapsulations sont possibles,
- Tout protocole encapsulé doit être désencapsulé plus loin dans le chemin, et dans un ordre particulier correspondant aux encapsulations,
- Une désencapsulation ne peut avoir lieu que si l'encapsulation qui lui correspond a été faite avant. A chaque désencapsulation, il faut connaître le protocole courant, mais également le protocole encapsulé, et il faut que la désencapsulation correspondante soit disponible à cet endroit.

Nous dirons qu'un chemin est *faisable* s'il respecte ces contraintes. Calculer un tel chemin n'est pas trivial. Il a été montré que le problème, avec une contrainte de bande passante, est NP-complet. Le statut du problème sans contrainte de QoS n'était pas connu.

1.6.2.2 Contributions de la thèse sur le calcul de chemins avec prise en compte des encapsulations

Dans le chapitre 4, nous rappelons la structure en couches des réseaux et l'hétérogénéité protocolaire qui les caractérise. Nous présentons en détail certains exemples d'encapsulations ainsi que l'architecture dans laquelle elles sont définies. Enfin, nous montrons plusieurs cas concrets où le calcul de chemins avec prise en compte des encapsulations se pose.

Notre but et de proposer un algorithme de calcul de chemins avec QoS, mais il se trouve que ce problème est très complexe même sans QoS. Dans le chapitre 5, nous étudions donc d'abord le problème de la prise en compte des encapsulations (que nous appellerons *calcul de chemin dans un réseau multicouche* dans la suite de cette thèse) sans QoS et nous voyons que le problème peut s'exprimer avec des outils de Théorie des Langages. En effet, un réseau multicouche avec encapsulations se modélise très naturellement en automate à pile où les protocoles représentent un alphabet. De plus, la suite de protocoles utilisés dans un chemin faisable est un mot accepté par l'automate. Inversement, si nous trouvons un mot (suite de protocoles) accepté par l'automate à pile, nous pouvons en déduire un chemin faisable. Le

mot le plus court accepté par l'automate correspond au chemin faisable le plus court dans un réseau multicouche. Nous utilisons donc des outils de Théorie des Langages pour générer ce mot, et calculons le chemin correspondant. Nous montrons la correction de tous nos algorithmes ainsi que leur complexité polynomiale. Notre solution est donc la première solution polynomiale au problème de calcul de chemins dans un réseau multicouche. Elle n'est possible que sans contraintes de QoS.

La NP-complétude du problème est connue si on introduit une contrainte de bande passante. Nous améliorons ce résultat en montrant que le problème est NP-complet même s'il n'y a que deux protocoles. Enfin, nous nous intéressons au problème avec plusieurs contraintes de QoS. Il existe des algorithmes pour calculer des chemins sous contraintes de QoS mais sans prise en compte des encapsulations. L'un d'entre eux, bien qu'exponentiel dans le pire des cas, a un temps d'exécution raisonnable en moyenne. Nous l'adaptons donc pour qu'il prenne en compte également les encapsulations.

1.7 Conclusion

Face à l'augmentation exponentielle du trafic sur les réseaux (particulièrement sur Internet) et les applications de plus en plus gourmandes en ressources, les opérateurs ne peuvent plus se contenter du surdimensionnement pour assurer une bonne qualité d'expérience à leurs clients et aux utilisateurs finaux. Il y a deux raisons à cela : le surdimensionnement n'assure aucune garantie dans le pire des cas, alors que certaines applications ont besoin de cette garantie ; et l'augmentation continuelle de la capacité des réseaux (qui est la principale ressource, et la plus contrainte) se ralentit et ne contrebalance plus l'augmentation du trafic.

Une gestion plus intelligente des ressources réseau est nécessaire. Une solution possible est de définir des garanties de QoS et les offrir en tant que produit à valeur ajoutée. Cette solution permettrait d'optimiser l'utilisation des ressources, mais également de créer une nouvelle source de revenus pour les opérateurs. En effet, la plus grande partie des revenus générés par le réseau est récupérée par les fournisseurs de services et de contenus. Alors que la plus grande partie des investissements est assurée par les opérateurs et les domaines de transit afin de fournir une capacité suffisante. La QoS monnayé peut leur assurer un retour sur investissement et ainsi les inciter à améliorer la capacité des réseaux.

Le déploiement de la QoS au sein d'un même domaine est techniquement possible. Plusieurs mécanismes sont disponibles pour l'assurer. Mais dans un contexte inter-domaine, ce déploiement s'avère beaucoup plus complexe. La principale difficulté étant les intérêts économiques divergents des domaines, dont la conséquence est une absence de coopération technique. Or, assurer la QoS est impossible sans coopération. Le projet européen ETICS a défini plusieurs modèles possibles pour une coopération entre les domaines afin de permettre le déploiement d'une QoS en inter-domaine. Différentes solutions sont proposées selon le degré de confiance et l'implication des différents domaines. Ces solutions se basent sur la négociation de SLA, en centralisé ou en distribué et selon plusieurs scénarios. C'est dans ce cadre que les contributions de la thèse se situent. La première étant de proposer des méthodes de calcul de stratégies optimales pour les domaines qui fournissent les SLA, que ce soit de manière exacte ou en utilisant des algorithmes d'apprentissage comme dans le chapitre 2, ou bien en adaptant les méthodes d'apprentissage à la prise en compte de la réputation des domaines comme dans le chapitre 3. Nous avons également proposé des algorithmes polynomiaux pour le calcul de chemins dans des réseaux hétérogènes où les domaines utilisent différents protocoles et où il faut prendre en compte l'encapsulation de certains protocoles dans d'autres. Ces solutions seront détaillées dans les chapitres 4 et 5.

Après avoir défini la QoS et présenté certains mécanismes de déploiement en intra-domaine, nous avons rappelé dans ce chapitre le fonctionnement des réseaux inter-domaine (principalement Internet) aujourd'hui, aussi bien sur le plan économique que technique et protocolaire. Nous avons expliqué les difficultés de déploiement de la QoS en inter-domaine et les possibles solutions proposées par le projet ETICS. Enfin, nous avons placé les contributions de cette thèse dans ce contexte. Les chapitres suivants détaillerons ces contributions.

Première partie

Négociation de SLA dans les réseaux inter-domaine

2 Négociation de SLA

2.1 Introduction

Dans la première partie de cette thèse, nous nous intéressons au problème de la négociation de SLA du point de vue du domaine fournisseur. Cela consiste pour lui à choisir la meilleure offre à faire en réponse à une requête de QoS venant d'un client, meilleure dans le sens où elle maximise les revenus à long terme du domaine fournisseur. Plusieurs paramètres sont à prendre en compte : le prix du SLA, sa probabilité d'échec, les offres des concurrents, la fonction de choix du client, etc. Ces paramètres ne sont pas forcément indépendants, ce qui rend le problème plus complexe. De plus, le choix d'une offre à l'instant présent peut influer sur les paramètres des négociations futures (typiquement, le provisionnement d'un SLA et sa consommation de ressources influent sur la probabilité d'échec des SLA futurs). Dès lors, comment déterminer une stratégie qui maximise les revenus à long terme du ou des domaines fournisseurs ?

Les situations où le fournisseur est en monopole et la situation où plusieurs fournisseurs sont en concurrence n'est pas la même. Dans le premier cas, il s'agit d'un problème d'optimisation, sachant que le client n'a pas le choix car il ne reçoit qu'une seule offre. Dans le second cas, les stratégies optimales sont différentes si les fournisseurs sont égoïstes où s'ils collaborent.

En posant des hypothèses simples sur certains paramètres (durée et prix des SLA, état des ressources, probabilités d'échec), le choix de stratégie peut se modéliser en un problème d'optimisation classique : le problème du cycle de poids moyen maximum. Trouver un tel cycle dans le graphe des états du fournisseur permet de trouver une stratégie optimale. Cette méthode permet également de déterminer les stratégies optimales de plusieurs fournisseurs s'ils collaborent. Si les fournisseurs sont égoïstes, les stratégies optimales peuvent générer moins de revenus qu'en cas de collaboration. Nous montrons que la différence peut être arbitrairement grande.

Les méthodes citées ci-dessus présentent néanmoins plusieurs inconvénients : elles ne sont pas scalables pour un grand nombre de SLA et elles nécessitent de connaître parfaitement tous les paramètres du réseau. Elles ne peuvent donc pas être implémentées à grande échelle. Une solution ne nécessitant pas de connaître tous ces paramètres et passant à l'échelle est l'utilisation d'algorithmes d'apprentissage. Ces algorithmes ne se basent que sur l'historique des négociations de SLA pour « apprendre » quelle offre faire selon les situations. Nous avons adapté et implémenté trois algorithmes d'apprentissage (*Learning Reward Inaction*, Q-Learning et SARSA) avec différentes politiques. Les simulations montrent que face à un client qui n'apprend pas, ces algorithmes convergent très vite vers la stratégie optimale. Dans le cas où deux fournisseurs concurrents utilisent les algorithmes d'apprentissage, quelques politiques se montrent meilleures que d'autres mais il n'y a pas d'ordre total concernant l'efficacité.

Ce chapitre est organisé de la manière suivante : la section 2.2 rappelle le problème ainsi que quelques travaux antérieurs dans le domaine. La section 2.3 présente le modèle que nous utilisons pour la négociation de SLA. La section 2.4 étudie le cas d'un client et d'un fournisseur en situation de monopole. La section 2.5 étudie le cas où deux fournisseurs concurrents collaborent et le cas où ils sont égoïstes. La section 2.6 discute des limites des méthodes utilisées. La section 2.7 présente les différents algorithmes d'apprentissage et leur adaptation au problème de négociation de SLA. La section 2.8 compare ces algorithmes grâce à des résultats de simulations. Enfin, la section 2.9 conclut le chapitre.

Ce chapitre est basé sur les articles [C1] et [C3] publiés dans le cadre de cette thèse.

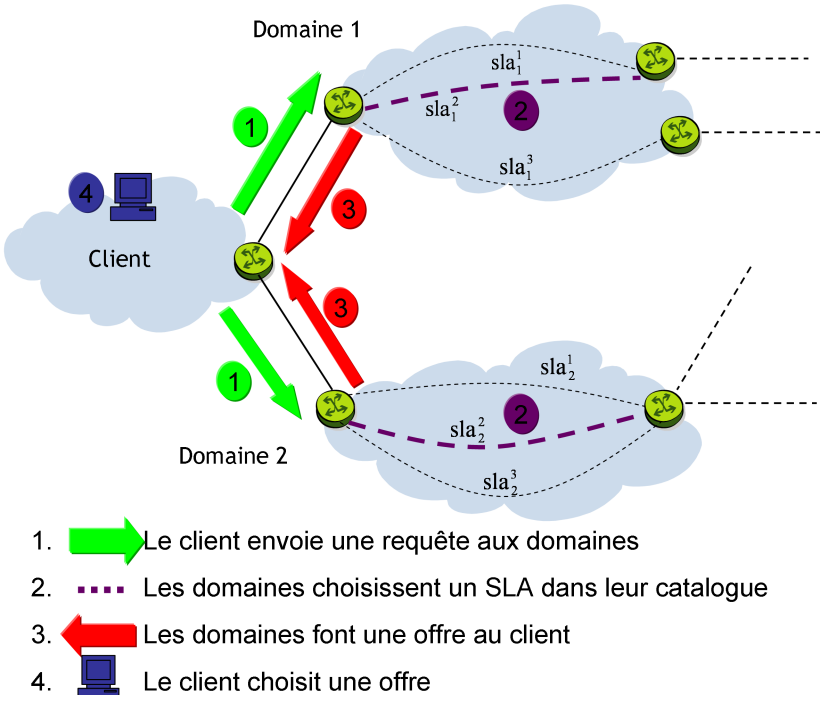


FIGURE 2.1 – Processus de négociation de SLA.

2.2 Négociation de SLA

2.2.1 Définition du problème

Rappelons que nous nous intéressons au problème de la négociation de SLA du point de vue du domaine fournisseur de SLA en nous plaçant dans le scénario suivant : le client envoie une requête demandant une certaine QoS pour une durée déterminée, le domaine fournisseur choisit une offre parmi ses SLA pré-calculés et l'envoie, ainsi que son prix, au client. Si le client reçoit plusieurs offres de différents domaines fournisseurs, il en choisit une et en informe le fournisseur. Si le client ne reçoit qu'une seule offre, il l'accepte forcément, le fournisseur est en situation de monopole. Le SLA est ensuite instancié. La figure 2.1, déjà présentée en chapitre 1 et replacée ici par commodité, illustre ce processus.

Ce processus peut se placer dans plusieurs scénarios envisagés par ETICS :

- Dans un scénario *pull* distribué, et particulièrement dans un processus en cascade, un domaine est d'abord fournisseur du domaine qui le précède, puis à son tour client du domaine qui le suit.
- Dans un scénario *push* totalement centralisé, le rôle du client sera tenu par l'entité neutre qui envoie une requête aux domaines. Chacun d'eux choisit un SLA et le propose à l'entité neutre.
- Dans un scénario *push* centralisé par domaine, le processus de négociation de SLA se produit une seule fois entre le client et le domaine voisin qui agrège les offres.

Bien que placé dans différents scénarios, le processus de négociation de SLA est le même. Comme les domaines peuvent être des clients et des fournisseurs, nous distinguerons le domaine fournisseur en l'appelant *Network Service Provider* (NSP) dans la suite de cette thèse.

2.2.2 Négociation de SLA et prise en compte du risque d'échec

Le problème de négociation de SLA et la prise en compte du risque d'échec dans l'allocation de ressources sont deux problèmes généralement traités séparément. Dans [53], Groléat Et Pouyllau proposent l'utilisation d'algorithmes d'apprentissage pour la négociation de SLA. Suksomboon *et al.* [109] analysent le problème de négociation de SLA par une approche basée sur la théorie de jeux. Ils spécifient les stratégies qui permettent d'atteindre un équilibre dans certains cas.

En ce qui concerne la prise en compte du risque d'échec dans l'allocation de ressources, Xia *et al.* [117] proposent une stratégie de provisionnement qui minimise le risque de violation de SLA et permet le

calcul de chemins à risque minimum dans les réseaux WDM [14]. Les auteurs stipulent que la prise en compte statistique de la disponibilité d'un chemin n'est pas suffisante pour minimiser le risque d'échec. Ils proposent de prendre en compte des paramètres plus pertinents comme les différents types d'échecs, le seuil de disponibilité souhaité, etc.

Ces travaux ne font pas la relation entre la négociation de SLA (compétition pour être sélectionné par le client) et stratégie de provisionnement à long terme (comment minimiser le risque d'échec des SLA). En modélisant la probabilité d'échec par une fonction des ressources déjà provisionnées, la négociation de SLA peut se voir comme un problème d'optimisation.

2.3 Modèle des NSP et des clients

Dans cette section, nous nous proposons de modéliser certains paramètres clés de la négociation de SLA. Ce modèle servira de base aux algorithmes proposés et aux simulations.

2.3.1 Caractéristiques des NSP

Nous supposons que chaque NSP a une capacité maximale c_{max}. Soit c_t la capacité utilisée d'un NSP entre l'instant $t-1$ et l'instant t (le temps est discrétisé). Nous définissons le taux de capacité utilisée à l'instant t comme étant $\rho_t = c_t/c_{\max}$ et le taux moyen de capacité utilisée (sur une période quelconque T) comme étant $\tilde{\rho} = (\sum_{t=1}^{T} \rho_t)/T$.

Un SLA proposé par un NSP i est noté q_i^j. C'est un triplet (b_i^j, d_i^j, ℓ_i^j) où b_i^j est la bande passante, d_i^j le délai et ℓ_i^j le taux de perte (de paquets).

Par souci de simplicité, nous supposons que tous les SLA ont la même durée, notée Δ. Autoriser des durées différentes est possible mais cela complique la définition des états d'un NSP. L'ensemble des SLA dont dispose le NSP i est noté $\mathcal{Q}_i$, ou simplement $\mathcal{Q}$ quand il ne peut y avoir de confusion sur l'identité du NSP fournisseur.

2.3.1.1 Prix unitaire des SLA

Chaque NSP fixe un prix unitaire noté p_u. Donc le prix d'un SLA q_i^j par unité de temps est de $p_u \cdot b_i^j$ prix total d'un SLA est de $p_u \cdot b_i^j \cdot \Delta$.

2.3.1.2 La fonction d'échec

La probabilité d'échec d'un SLA dépend du taux de bande passante utilisée par le NSP. C'est une fonction du taux de capacité utilisée et est notée $f(\rho_t)$. La fonction d'échec peut être $f(\rho_t) = \rho_t$, $f(\rho_t) = \rho_t^2$ ou encore $f(\rho_t) = 1 - \exp(-\rho_t)$.

2.3.1.3 Gains du NSP

Un NSP obtient un gain à chaque instant t. Le gain obtenu à l'instant t est noté v_t. Si c_t est la capacité utilisée à l'instant t alors :

$$v_t = \begin{cases} p_u \cdot c_t = p_u \cdot c_{\max} \cdot \rho_t & \text{s'il n'y a pas d'échec à l'instant } t \\ 0 & \text{sinon} \end{cases}$$

Donc le gain espéré à l'instant t est $\mathbb{E}(v_t) = p_u \cdot c_{\max} \cdot \rho_t \cdot (1 - f(\rho_t))$.

2.3.2 Caractéristiques des clients

Pour pouvoir comparer les différentes sensibilités des clients à la QoS et au prix des SLA, les paramètres de QoS sont regroupés dans une mesure normalisée dans [0,1[.

14. WDM : Wavelength Division Multiplexing.

2.3.2.1 Mesure de QoS

Une mesure $||\cdot||$ de la QoS fournie par un SLA est calculée en fonction des paramètres de ce SLA. Cette mesure est définie de la manière suivante :

$$||\mathrm{q}_i^j|| = 1 - \frac{1}{3}\left(\frac{\ell_i^j}{\ell_{\mathrm{req}}} + \frac{d_i^j}{d_{\mathrm{req}}} + \frac{b_{\mathrm{req}}}{b_i^j}\right),$$

où b_{req} est la bande passante minimum demandée par le client, d_{req} est le délai maximum demandé par le client et ℓ_{req} le taux de perte maximum toléré par le client. Si le SLA q_i^j satisfait ces contraintes de QoS, alors $||\mathrm{q}_i|| \in [0,1[$. Un SLA qui satisfait exactement les paramètres de la requête aura une mesure de 0. La fonction $||.||$ tend vers 1 quand les paramètres de QoS du SLA dépassent largement les contraintes du client. Les SLA qui ne satisfont pas la requête du client ne sont pas considérés.

2.3.2.2 L'utilité d'un client en fonction d'un SLA

Nous avons opté pour l'utilité classique (QoS − prix) :

$$U(\mathrm{q}_i^j) = ||\mathrm{q}_i^j|| - \eta\frac{p_i^j}{p_{\max}} \tag{2.1}$$

où η est un paramètre fixé qui dépend du client, p_i^j est le prix du SLA q_i^j, et $p_{\max}$est le prix maximum que le client est prêt à payer. Le paramètre η pondère l'influence du prix sur la fonction d'utilité du client.

2.4 Le cas d'un NSP fournisseur et d'un client

Dans cette section, nous nous intéressons au cas de figure d'un NSP fournisseur (que l'on nommera NSP 1) ayant le monopole pour répondre aux requêtes d'un client. Les SLA sont simplement notés q^j et leurs paramètres sont notés b^j, d^j et ℓ^j car il n'y a pas de confusion possible sur l'identité du NSP fournisseur.

2.4.1 L'ensemble d'états du NSP fournisseur

Nous nous proposons de définir formellement l'ensemble d'états du NSP 1. Pour pouvoir offrir le SLA qui maximise son gain moyen, le NSP 1 a besoin d'une information complète sur sa capacité disponible actuelle ainsi que sa capacité disponible future. A chaque instant t, le NSP 1 connaît les SLA qu'il a provisionnés jusqu'à t. Les SLA provisionnés avant $t-\Delta$ sont déjà terminés, mais les SLA provisionnés entre $t-\Delta$ et t utilisent encore la capacité du NSP. Ces SLA permettent donc de savoir quelle capacité sera libérée à chaque instant de décision entre t et $t+\Delta$.

Nous définissons donc l'*état* du NSP 1 à l'époque de décision t comme étant la liste ordonnée de SLA qui seront libérés entre les époques de décision t et $t+\Delta$. En fait, le NSP doit seulement connaître leur bande passante. Plus formellement, l'état du NSP 1 à l'époque de décision t est un vecteur $s_t = (b^1,\ldots,b^\Delta)$ où b^j est la bande passante du SLA qui sera libéré à l'époque de décision $t+j$. La capacité utilisée du NSP à l'instant t est donc $c_t = \sum_{i=1}^{\Delta} b^j$.

Clairement, si la capacité totale du NSP 1, notée $c_{\max}$, n'est pas bornée, alors le nombre total d'états possibles est $(|\mathcal{Q}|+1)^\Delta$. Cependant, puisque $\forall t$, $c_t \leq c_{\max}$, les états $s' = (b'^1,\ldots,b'^\Delta)$ qui satisfont $\sum_{j=1}^{\Delta} b^j > c_{\max}$ ne sont pas valides, car pour atteindre ces états il faut utiliser une capacité supérieure à la capacité totale du NSP 1. Soit $\mathcal{S}_1$ l'ensemble des états valides (ou atteignables) du NSP 1. Alors

$$(|\mathcal{Q}|+1)^{\min(\Delta,K)} \leq |\mathcal{S}_1| \leq (|\mathcal{Q}|+1)^{\min(\Delta,K')},$$

où $K = \lfloor c_{\max} / \max_{\mathrm{q}^j \in \mathcal{Q}_1} b^j \rfloor$ et $K' = \lfloor c_{\max} / \min_{\mathrm{q}^j \in \mathcal{Q}_1} b^j \rfloor$.

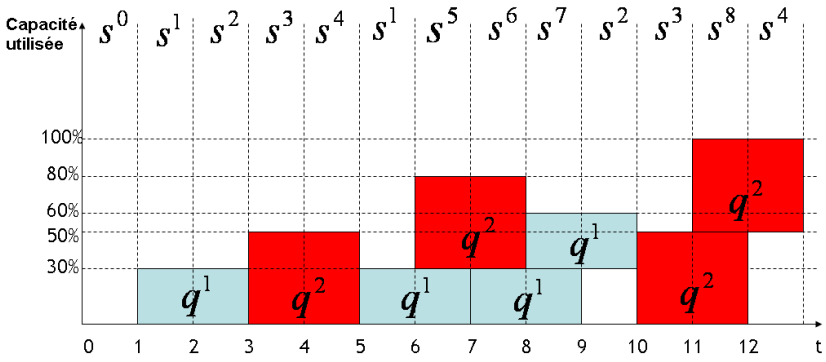


FIGURE 2.2 – Un diagramme montrant les différents états du NSP 1.

Exemple. Cet exemple montre comment les états d'un NSP sont définis en fonction de l'ensemble de ses SLA. Soit $c_{\max} = 100$Gbps la capacité maximum du NSP 1. Et soit $\{q^1,q^2\}$ son ensemble de SLA, tel que $b^1 = 30$Gbps$= 30\%.c_{\max}$ et $b^2 = 50$Gbps$= 50\%.c_{\max}$. La durée des SLA est $\Delta = 2$ unités de temps. Le diagramme de la figure 2.2 montre les différents états possibles du NSP 1. Toute combinaison (et chevauchement) de SLA définit un état. Par exemple, $s^5 = (b^1,b^2)$ est l'état où q^1 sera libéré à l'instant de décision $t+1$ et q^2 sera libéré à l'instant $t+2$. Dans cet exemple, $|\mathcal{S}_1| = (|\mathcal{Q}_1|+1)^{\Delta} = (2+1)^2 = 9$. Tous les états sont valides.

Remarque. L'exemple précédent, tout comme certains exemples qui suivent, n'est évidemment pas réaliste. Le but de cet exemple est d'illustrer l'approche et les méthodes utilisées dans ce chapitre plutôt que de fournir des données réalistes. Dans un cas réel, les SLA ont une granularité beaucoup plus fine (principalement moins de 5% de la capacité totale d'un NSP) et une durée bien plus longue. Un exemple réaliste ne conviendrait pas pour expliquer la méthode utilisée car le nombre d'états serait trop grand et ne pourrait pas être représenté convenablement. Les simulations seront effectuées sur des cas plus réalistes.

2.4.2 Maximiser le gain moyen du NSP

Dans cette section, nous cherchons à calculer la stratégie optimale à long terme du NSP 1, c'est-à-dire la séquence de SLA à offrir à chaque instant t pour maximiser l'espérance de son gain moyen. Pour cela, nous allons modéliser l'ensemble de ses états et les transitions entre ces états comme étant une matrice $(\max,+)$. Nous présenterons d'abord brièvement les algèbres et matrices $(\max,+)$, puis nous verrons comment ces outils modélisent l'ensemble d'états du NSP. Enfin, un calcul du cycle de poids maximum dans ce modèle nous permettra de dériver la stratégie optimale du NSP 1.

2.4.2.1 Algèbres et matrices $(\max,+)$

Considérons le semi-anneau $\mathbb{R}_{\max} \stackrel{\text{def}}{=} (\mathbb{R} \cup \{-\infty\}, \max, +)$. Suivant la notation standard dans les ouvrages sur le sujet (notamment [19]) pour chaque $a,b,a_i \in \mathbb{R} \cup \{-\infty\}$, nous noterons $-\infty$ par le caractère ε, $\max(a,b)$ par $a \oplus b$, $\max_i a_i$ par $\bigoplus_i a_i$, $a+b$ par $a \otimes b$ et $\sum_i a_i$ par $\bigotimes_i a_i$.

Soit $A = (A_{ij})_{1\le i\le n, 1\le j\le n}$ une matrice carrée avec des éléments dans $\mathbb{R}_{\max}$. Nous redéfinissons le carré de la matrice A, noté $A^2 = A \otimes A$, comme suit :

$$(A^2)_{ij} = \bigoplus_{l=1}^{n} A_{il} \otimes A_{lj} \tag{2.2}$$

Et la m-ième puissance de A comme suit :

$$(A^m)_{ij} = \bigoplus_{l=1}^{n} (A^{m-1})_{il} \otimes A_{lj} \tag{2.3}$$

Pour plus de détails sur les algèbres et les matrices $(\max,+)$, le lecteur intéressé peut consulter l'ouvrage de référence de Baccelli *et al.* [19].

2.4.2.2 L'ensemble des transitions entre états du NSP

La transition d'un état s^t vers un état s^{t+1} se fait en libérant le SLA provisionné à l'instant $t-\Delta$ (s'il existe) et en provisionnant le SLA offert à l'instant t (s'il existe aussi). Cette transition est valuée par l'espérance de gain du NSP 1, c'est-à-dire $v_{t+1}\cdot(1-f(\rho_{t+1}))$. Ainsi, nous définissons la fonction de transition $\mathcal{T}:\mathcal{S}\cdot\mathcal{S}\to\mathbb{R}$ par $\mathcal{T}(s^t,s^{t+1})=v_{t+1}\cdot(1-f(\rho_{t+1}))$. L'ensemble des états valides $\mathcal{S}$ associé à l'ensemble des transitions $\mathcal{T}$ peut être vu comme un graphe pondéré où $\mathcal{S}$ est l'ensemble des nœuds et il y a un arc de poids w de $s^i\in\mathcal{S}$ vers $s^j\in\mathcal{S}$ si et seulement si $\mathcal{T}(s^i,s^j)=w$. Ce graphe peut être représenté par sa matrice d'incidence pondérée $A=(A_{i,j})_{1\leq i\leq|\mathcal{S}|,1\leq j\leq|\mathcal{S}|}$ où :

$$A_{ij}=\begin{cases}\mathcal{T}(s^j,s^i) & \text{s'il y a une transition de } s^j \text{ vers } s^i\\ \varepsilon & \text{sinon}\end{cases} \tag{2.4}$$

Remarque. Il est à noter que A_{ij} est le poids de la transition de s^j vers s^i. Ceci est dû au fait que les équations matricielles dans $\mathbb{R}_{\max}$ sont traditionnellement écrites en utilisant des vecteurs colonnes.

La matrice A est une matrice $(\max,+)$. Le but du NSP 1 est de trouver un cycle $s^1,s^2,\dots s^i$ dans le graphe des états qui maximise l'espérance du gain moyen. Ce problème correspond au problème bien connu du cycle à moyenne maximum [15].

2.4.2.3 Le cycle de poids moyen maximum

Selon le théorème spectral des matrices $(\max,+)$ [19], l'espérance maximum du gain moyen peut-être obtenue par la formule :

$$\mathbb{E}(v_{max})=\max_{i=1..|\mathcal{S}|}\frac{\max_{k=1..|\mathcal{S}|}(A^i)_{kk}}{i} \tag{2.5}$$

L'intuition derrière cette formule est très simple : l'élément $(A^i)_{kk}$ est le poids maximum d'un cycle de longueur i partant du nœud s^k et revenant à celui-ci (ce qui est équivalent à dire que c'est le cycle de poids maximum de longueur i qui passe par l'état s^k). Il suffit de calculer toutes les puissances A^i et chercher à chaque fois la plus grande valeur.

La formule (2.5) suggère un algorithme naïf de calcul de $\mathbb{E}(v_{\max})$ en $O(|\mathcal{S}|^4)$. En effet, il suffit de calculer $A^i=A\otimes A^{i-1}$ pour toutes les valeurs de $i=2\dots|\mathcal{S}|$ (ce qui prend $O(|\mathcal{S}|^2)$ opérations), et de chercher l'élément le plus grand dans chaque matrice A^i (ce qui prend également $O(|\mathcal{S}|^2)$).

Il est possible de diminuer cette complexité en utilisant une caractérisation du cycle de poids moyen maximum due à Karp [64].

Théorème 1 (Karp [64])**.** *Soit s^j un état quelconque tel que $j\in\{1,\dots,|\mathcal{S}|\}$. Alors :*

$$\mathbb{E}(v_{max})=\max_{i=1..|\mathcal{S}|}\min_{k=0..|\mathcal{S}|-1}\frac{(A^{|\mathcal{S}|})_{ij}-(A^k)_{ij}}{|\mathcal{S}|-k}$$

Cette formule permet de déduire un algorithme de calcul de cycle de poids moyen maximum en $O(|\mathcal{S}|^3)$.

Exemple. Cette exemple montre le graphe correspondant aux états et transitions du NSP ainsi que le cycle de poids moyen qui maximise l'espérance de gain du NSP. Considérons le NSP 1 ayant les mêmes paramètres que dans l'exemple précédent. Son ensemble de stratégies (et de SLA) est donc $\{q^1,q^2\}$. Soit la fonction d'échec $f(\rho)=\rho$, $\rho\in[0,1]$ (la fonction d'échec est linéaire par rapport à la charge). Et soit le prix par unité $p_u=1$ unité monétaire par Gbps. La figure 2.3 montre l'ensemble des états et des transitions associés au NSP 1. Les transitions sont valuées par l'espérance de gain du NSP en atteignant l'état d'arrivée. Le cycle de poids moyen maximum est $s^3,s^4,s^3\dots$ L'espérance de gain moyen maximum est de 25 et le taux de capacité utilisée optimal est $\rho_{\text{opt}}=50\%$.

15. *Maximum mean cycle problem* en anglais.

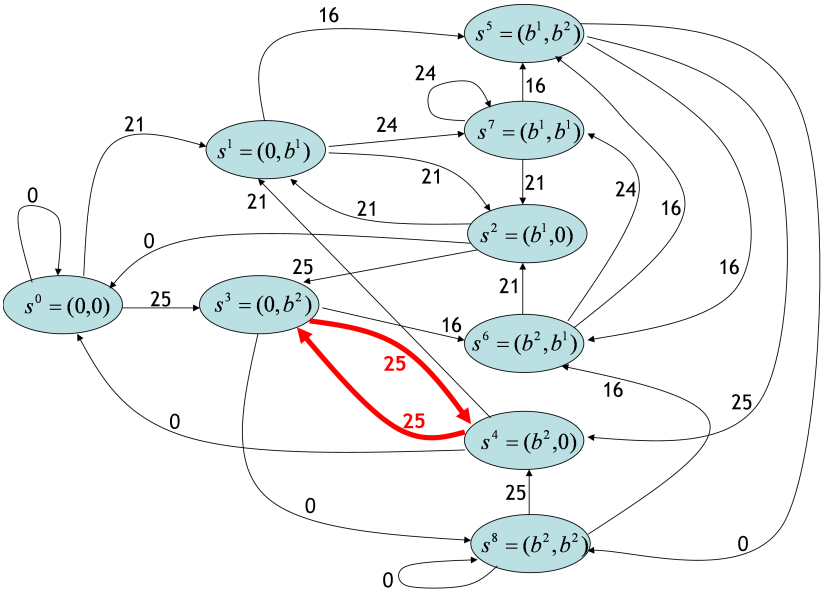


FIGURE 2.3 – Le graphe d'états et de transitions associé au NSP 1. Chaque b^j est une bande passante de SLA. Le cycle en rouge est le cycle de poids moyen maximum.

2.4.3 Une autre approche pour le calcul de la stratégie optimale du NSP

Supposons que le NSP 1 peut fixer sa capacité utilisée moyenne $\tilde{\rho} = \tilde{c}/c_{\max}$ (où $\tilde{c}$ est la capacité utilisée moyenne) à n'importe quelle valeur dans [0,1]. Et supposons qu'il connaît la valeur de la fonction d'échec $f(\tilde{\rho})$ pour chaque $\tilde{\rho} \in [0,1]$. Le gain du NSP 1 dépend de la bande passante des SLA provisionnés :

$$v = p_u \cdot c_{\max} \cdot \tilde{\rho}$$

Le gain moyen par unité de temps est donné par $v(1 - f(\tilde{\rho}))$. Le but du NSP 1 est de calculer la capacité utilisée optimale, qui est donnée par $\rho_{\text{opt}} = \arg\max_{\tilde{\rho} \in [0,1]} v(1 - f(\tilde{\rho}))$.

Si le NSP 1 connaît les valeurs de $f(\tilde{\rho})$ et v pour chaque $\tilde{\rho} \in [0,1]$, alors il est facile de calculer la capacité utilisée moyenne optimale, c'est-à-dire qui maximise l'espérance du gain moyen. Cette approche est utile uniquement dans le cas où les SLA (et leur bande passante) ont une granularité fine, ce qui permet d'approximer finement ρ_{opt} par $(\sum_{t=1}^{\Delta} c_t)/\Delta$.

Exemple. Soient, comme dans l'exemple précédent, la capacité maximum du NSP 1 $c_{\max} = 100$Gbps, le prix unité $p_u = 1$ u.m. et la fonction d'échec $f(\tilde{\rho}) = \tilde{\rho}$. Le gain moyen maximum $\mathbb{E}(v_{\max})$ est obtenu en maximisant la fonction $v(1 - f(\rho))$. La valeur maximum est obtenue quand $\rho = 0.5$. L'espérance du gain moyen maximum est $\mathbb{E}(v_{\max}) = 25$u.m. et la capacité utilisée moyenne optimale est $\rho_{\text{opt}} = 50\%$. Cette valeur correspond exactement au cycle de poids moyen maximum obtenu dans l'exemple précédent.

2.5 Le cas de deux NSP fournisseurs et d'un client

Dans cette section, nous étudions la négociation de SLA entre un client et deux NSP fournisseurs (NSP 1 et NSP 2). Nous supposons que le client envoie une requête à chaque instant au deux NSP et que les deux NSP répondent chacun par une offre de SLA ou une absence d'offre. Si un seul des deux NSP offre un SLA, le client sélectionne forcément cette offre. Si les deux NSP font chacun une offre, le NSP choisira parmi ces deux offres. Une seule offre peut être sélectionnée à chaque instant t. Si l'offre du NSP 1 est sélectionnée, alors le NSP 2 ne provisionnera aucun nouveau SLA, et inversement. Si les deux offres sont similaires (par rapport à la fonction d'utilité du client), alors le client sélectionne une des offres aléatoirement (et uniformément).

2.5.1 Stratégie optimale des deux NSP s'ils collaborent

Dans cette section, les deux NSP sont considérés comme formant un seul système dont le but et de maximiser la somme des gains des deux NSP. L'approche pour calculer le gain maximum et les stratégies optimales est la même qu'en section 2.4.2.

Soit $\mathcal{S}_1$ (resp. $\mathcal{S}_2$) l'ensemble des états valides du NSP 1 (resp. NSP 2) comme défini dans la section 2.4.1. Et soit $\mathcal{T}_1$ (resp. $\mathcal{T}_2$) la fonction de transition du NSP 1 (resp. NSP 2) comme définie en section 2.4.2.2. Pour chaque état $s_i^j = (b_i^1,\ldots,b_i^\Delta) \in \mathcal{S}_i$ (où $i \in \{1,2\}$), nous définissons $suc_\emptyset(s_i^j)$ comme étant l'état $suc_\emptyset(s_i^j) = (b_i^2,b_i^3,\ldots,b_i^\Delta,0)$. L'état $suc_\emptyset(s_i^j)$ est le successeur de l'état s_i^j si le NSP i ne provisionne aucun SLA à l'instant présent. Clairement, si $s_i^j \in \mathcal{S}_i$ alors $suc_\emptyset(s_i^j) \in \mathcal{S}_i$ puisque la capacité utilisée correspondant à l'état $suc_\emptyset(s_i^j)$ est inférieure à la capacité utilisée à l'état s_i^j. Si l'état s_i^j est valide, l'état $suc_\emptyset(s_i^j)$ l'est aussi.

L'ensemble des états globaux du système, noté $\mathcal{S}$, est inclus dans le produit cartésien $\mathcal{S}_1 \times \mathcal{S}_2$. Si $s = (s_1,s_2) \in \mathcal{S}_1 \times \mathcal{S}_2$ et $s' = (s'_1,s'_2) \in \mathcal{S}_1 \times \mathcal{S}_2$ alors une transition de l'état s vers l'état s' existe si et seulement si au moins une de ces trois conditions est satisfaite :

1. Il y a une transition de l'état s_1 vers l'état s'_1 et $s'_2 = suc_\emptyset(s_2)$;
2. Il y a une transition de l'état s_2 vers l'état s'_2 et $s'_1 = suc_\emptyset(s_1)$;
3. $s'_1 = suc_\emptyset(s_1)$ et $s'_2 = suc_\emptyset(s_2)$.

Il est à noter que le cas 3. est redondant par rapport à 1. et 2. puisque $(s_i,suc_\emptyset(s_i))$ est toujours une transition dans $\mathcal{T}_i$. L'ensemble des états globaux $\mathcal{S}$ est exactement l'ensemble des états dans la composante fortement connexe à laquelle appartient l'état $((0,\ldots,0),(0,\ldots,0))$ dans le graphe formé par le couple $(\mathcal{S},\mathcal{T})$. De plus, la fonction de transition $\mathcal{T}$ sur $\mathcal{S}$ est valuée de la façon suivante :

$$\mathcal{T}(s,s') = \mathcal{T}_1(s_1,s'_1) + \mathcal{T}_2(s_2,s'_2). \tag{2.6}$$

Définissons la matrice $A = (A_{l,j})_{1\leq l\leq|\mathcal{S}|,1\leq j\leq|\mathcal{S}|}$ de façon similaire à la section 2.4.2. C'est-à-dire :

$$A_{lj} = \begin{cases} \mathcal{T}(s^j,s^l) & \text{s'il y a une transition de } s^j \text{ vers } s^l \\ \varepsilon & \text{sinon} \end{cases} \tag{2.7}$$

De manière similaire au cas avec un seul NSP fournisseur, le gain moyen maximum de l'ensemble du système est donné par :

$$\mathbb{E}(v_{\max}) = \max_{i=1..|\mathcal{S}|} \frac{\max\limits_{k=1\ldots|\mathcal{S}|}(A^i)_{kk}}{i}$$

Cette valeur peut être calculée soit directement, soit par le théorème de Karp (théorème 1).

Proposition 1. *Soit $\mathbb{E}(v^1_{\max})$ (resp. $\mathbb{E}(v^2_{\max})$) le gain moyen maximum du NSP 1 (resp. NSP 2) dans le cas où le NSP est seul face à un client. Alors :*

$$\max(\mathbb{E}(v^1_{\max}),\mathbb{E}(v^2_{\max})) \leq \mathbb{E}(v_{\max}) \leq \mathbb{E}(v^1_{\max}) + \mathbb{E}(v^2_{\max}).$$

Démonstration. Pour $i \in \{1,2\}$, posons $s_i^0 = (0,\ldots,0)$. L'inégalité $\mathbb{E}(v^1_{\max}) \leq \mathbb{E}(v_{\max})$ se déduit directement du fait que $\mathcal{S}_1 \times \{s_2^0\} \subseteq \mathcal{S}$ et que si $\mathcal{T}_1(s_1^k,s_1^j) \neq -\infty$, alors par construction $\mathcal{T}((s_1^k,s_2^0),(s_1^j,s_2^0)) = \mathcal{T}_1(s_1^k,s_1^j) + \mathcal{T}_2(s_2^0,s_2^0) = \mathcal{T}_1(s_1^k,s_1^j)$. Par conséquent, pour chaque cycle dans $(\mathcal{S}_1,\mathcal{T}_1)$, il y a un cycle de même longueur et de même poids dans $(\mathcal{S},\mathcal{T})$ et $\mathbb{E}(v^1_{\max}) \leq \mathbb{E}(v_{\max})$. Par symétrie, inverser le rôle de $\mathbb{E}(v^1_{\max})$ et de $\mathbb{E}(v^2_{\max})$ est suffisant pour prouver que $\mathbb{E}(v^2_{\max}) \leq \mathbb{E}(v_{\max})$ et la première inégalité.

La deuxième inégalité est une conséquence directe du fait que $\mathcal{S} \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ et de l'équation (2.6) : à partir de n'importe quel cycle dans $(\mathcal{S},\mathcal{T})$ on peut construire deux cycles :

- De même longueur respectivement dans $(\mathcal{S}_1,\mathcal{T}_1)$ et $(\mathcal{S}_2,\mathcal{T}_2)$,
- Dont la somme des poids est égale au poids du cycle dans $(\mathcal{S},\mathcal{T})$.

Par conséquent, le poids moyen de ce cycle dans $(\mathcal{S},\mathcal{T})$ est la somme des poids moyens d'un cycle dans $(\mathcal{S}_1,\mathcal{T}_1)$ et d'un cycle dans $(\mathcal{S}_2,\mathcal{T}_2)$. □

2.5.2 Stratégie optimale quand les NSP sont égoïstes et Prix de l'Anarchie

Il y a deux possibilités quand les NSP sont égoïstes :

1. Chaque NSP a une stratégie dominante, c'est-à-dire que pour chaque NSP, il y a un SLA qui maximise l'espérance du gain moyen, indépendamment de la capacité utilisée actuelle et de l'état du NSP.
2. Un des deux NSP, ou aucun des deux, n'a de stratégie dominante.

Ces deux cas mènent à une différence dans le modèle mais donnent des résultats similaires.

2.5.2.1 Prix de l'Anarchie

Le Prix de l'Anarchie est défini comme étant le ratio entre le gain à l'équilibre de Nash (ou au pire équilibre quand il y en a plusieurs) et le gain maximum quand tous les joueurs collaborent. Intuitivement, il mesure la perte du système quand les joueurs sont égoïstes.

2.5.2.2 Cas où chaque NSP a une stratégie dominante

Nous supposons que chaque NSP a une stratégie dominante qui consiste à offrir un SLA particulier. Si un tel SLA existe, alors il est clairement celui qui maximise l'utilité du client. Offrir ce SLA à chaque instant de décision est une stratégie dominante pour les deux NSP. Soit q_1^* (resp. q_2^*) ce SLA pour le NSP 1 (resp. NSP 2). La configuration où chaque NSP i offre q_i^* est un équilibre de Nash du jeu [85] [16].

Si l'utilité du client par rapport au SLA q_1^* (resp. q_2^*) est plus grande que son utilité par rapport au SLA q_2^* (resp q_1^*), le client choisira toujours l'offre du NSP 1 (resp. NSP 2). Dans ce cas le NSP 1 domine le marché.

Si l'utilité du client par rapport aux SLA q_1^* et q_2^* est la même, nous supposons que le client choisit aléatoirement (avec une probabilité 1/2) entre q_1^* et q_2^*. Les différents états du système et les transitions entre ces états peuvent être modélisés comme une chaîne de Markov. Un état de cette chaîne de Markov est un couple $[s_1,s_2]$ où $s_1 \in \mathcal{S}_1$ et $s_2 \in \mathcal{S}_2$. Soit $suc_\emptyset(s_i)$ le successeur de l'état s_i tel que défini en section 2.5.1. Pour chaque état $s_i = (b_i^1,\ldots,b_i^\Delta)$, nous définissons $suc_{\mathrm{q}}(s_i)$ comme étant $suc_{\mathrm{q}}(s_i) = (b_i^2,\ldots,b_i^\Delta,b_i^*)$, où b_i^* est la bande passante correspondant au SLA q_i^*. En d'autres mots, $suc_\emptyset(s_i)$ est le successeur de l'état s_i si le NSP i ne provisionne aucun SLA à l'instant présent et $suc_{\mathrm{q}}(s_i)$ est le successeur de l'état s_i si le NSP i provisionne q_i^* à l'instant présent.

Tous les états récurrents de la chaîne de Markov sont de la forme $[(b_1^1,\ldots,b_1^\Delta),(b_2^1,\ldots,b_2^\Delta)]$ où $b_1^k = b_1^*$ et $b_2^k = 0$ ou $b_1^k = 0$ et $b_2^k = b_2^*$, puisque chaque NSP offre son « meilleur »SLA (stratégie dominante). Donc, à chaque instant le client sélectionne surement une des deux offres. Il s'ensuit que le nombre d'états récurrents de la chaîne de Markov est égal à 2^Δ.

Il y a une transition (avec une probabilité 1/2) de l'état $s = [s_1,s_2]$ vers l'état $s' = [s_1',s_2']$ si et seulement si :

$$\begin{cases} s_1' = suc_\emptyset(s_1) \text{ et } s_2' = suc_{\mathrm{q}}(s_2) \\ \text{ou} \\ s_1' = suc_{\mathrm{q}}(s_1) \text{ et } s_2' = suc_\emptyset(s_2) \end{cases}$$

Il est à noter que le Prix de l'Anarchie peut être arbitrairement grand en fonction de la différence de prix entre le SLA dominant (q_i^*) et le SLA optimal si le NSP était seul. L'exemple suivant montre un Prix de l'Anarchie supérieur à 1.

Exemple. Dans cet exemple, nous supposons que le NSP 1 et le NSP 2 ont la même capacité maximum $c_{\max} = 100$Gbps et le même ensemble de SLA : q^1 avec une bande passante de $b^1 = 30\text{Gbps} = 30\% \cdot c_{\max}$ et q^2 avec une bande passante $b^2 = 50\text{Gbps} = 50\% \cdot c_{\max}$. La durée de chaque SLA est $\Delta = 2$ unités de temps. La fonction d'échec est donnée par $f(\rho) = \rho$. Supposons que les requêtes du client sont telles que son utilité est maximisée par q^1. Donc offrir le SLA q^1 est une stratégie dominante pour les deux NSP et la configuration où les deux NSP offrent q^1 est un équilibre de Nash. Si la capacité utilisée d'un NSP est $c_t = 0$ ou $c_t = 30\% \cdot c_{\max}$, offrir q^1 plutôt que ne pas faire d'offre maximise le gain espéré du NSP (le gain espéré d'un NSP quand sa capacité utilisée est de $c_t = 60\% \cdot c_{\max}$ est supérieur au gain espéré

16. Rappelons qu'une configuration où chaque NSP joue sa stratégie dominante (propose son SLA q_i^*) est toujours un équilibre de Nash [90].

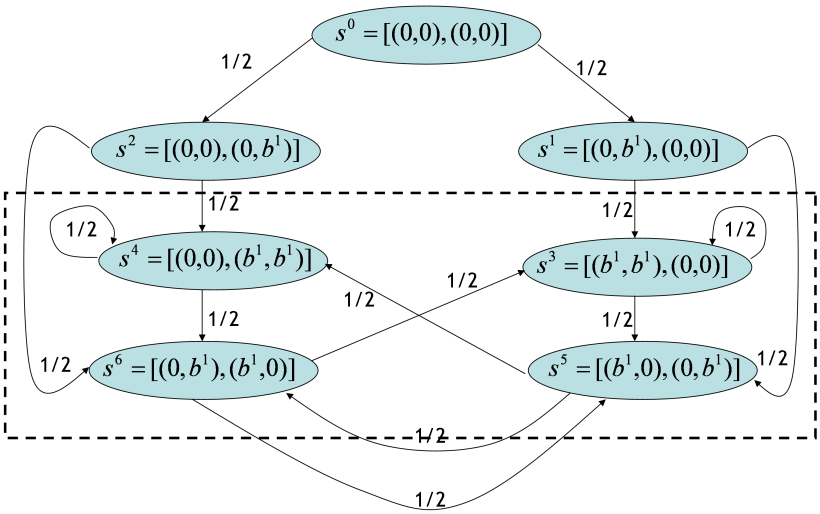


FIGURE 2.4 – La chaîne de Markov associée à l'exemple ci-dessus. Les états récurrents sont dans le rectangle en pointillé.

quand sa capacité est de $c_t = 30\% \cdot c_{\max}$). Nous supposons que si les deux NSP offrent le même SLA, le client choisira l'un d'eux avec une probabilité 1/2.

Il est clair que la stratégie optimale des NSP s'ils collaborent est d'offrir chacun alternativement q^2 puis ne pas faire d'offre (le NSP 1 offre q^2 à l'instant$t = 0$, puis le NSP 2 offre q^2 à l'instant $t = 1$, etc.). Le gain moyen espéré de chaque NSP est donc $\mathbb{E}(v_{\max}) = 25$ unités monétaires par unité de temps.

Le calcul du gain moyen espéré par chaque NSP dans la configuration d'équilibre de Nash décrite ci-dessus peut être fait comme suit : vu que les deux NSP offrent q^1 à chaque instant, et vu que le client choisit aléatoirement une des offres à chaque instant, les états du système peuvent être modélisés par une chaîne de Markov. La figure 2.4 montre la chaîne de Markov associée au système quand les deux NSP jouent leur stratégie dominante. Les états de la chaîne sont des combinaisons d'un état du NSP 1 et du NSP 2. Les arcs représentent les transitions entre les états et leur probabilité. Les états récurrents sont dans le rectangle en pointillé. La matrice de transition de la chaîne de Markov réduite (à ses états récurrents) est donnée par :

$$\begin{array}{c} \\ s^3 \\ s^4 \\ s^5 \\ s^6 \end{array} \begin{array}{c} \begin{array}{cccc} s^3 & s^4 & s^5 & s^6 \end{array} \\ \begin{pmatrix} 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 1/2 & 0 & 1/2 \\ 1/2 & 0 & 1/2 & 0 \end{pmatrix} \end{array}$$

Et la distribution stationnaire est de (1/4,1/4,1/4,1/4). Le gain moyen espéré de chaque NSP est donc :

$$\mathbb{E}(v_{\max}) = \frac{1}{4}60(1 - f(0{,}6)) + \frac{2}{4}30(1 - f(0{,}3)) = 16{,}5$$

Le Prix de l'Anarchie est donc de $25/16{,}5 \approx 1{,}51$.

Proposition 2. *Le Prix de l'Anarchie peut être arbitrairement grand en fonction des SLA et de leur prix.*

Démonstration. Comme vu dans l'exemple précédent, la stratégie dominante du NSP quand il est égoïste ne correspond pas forcément à sa stratégie optimale s'il collabore avec l'autre NSP. Il est facile de concevoir un jeu où le gain obtenus en cas de collaboration sont arbitrairement plus grands que les gains obtenus dans un équilibre de Nash. Il suffit de prendre les même paramètres que dans l'exemple précédent, mais de fixer la bande passante de q^1 aussi petite que l'ont veut. Son prix décroitra ainsi que le gain qui lui est associé, et le Prix de l'Anarchie sera aussi grand que l'ont veut. □

2.5.2.3 Absence de stratégie dominante

En fonction de son état, un NSP peut ne pas faire d'offre au lieu d'offrir un SLA en réponse à une requête du client (s'il a une bande passante utilisée trop élevée, par exemple). Dans ce cas, la chaîne de Markov qui modélise les états a une structure différente. En effet, le cas où les deux NSP n'offrent rien peut se présenter, ce qui amène à l'existence de transitions d'un état $s = [s_1,s_2]$ vers un état $s' = [s'_1,s'_2]$ où $s'_1 = suc_\emptyset(s_1)$ et $s'_2 = suc_\emptyset(s_2)$. Cependant, l'approche pour calculer le gain moyen espéré reste la même que précédemment.

2.6 Limitations des méthodes exactes pour la négociation de SLA

Dans le début de ce chapitre, nous avons vu que des méthodes d'optimisation (algèbres $(\max,+)$ et cycle de poids moyen maximum, chaînes de Markov, etc.) permettent de calculer des stratégies optimales dans le cas où un NSP est seul face à un client, sous l'hypothèse que les requêtes du client arrivent à intervalles réguliers et que la durée des SLA est toujours la même. Nous avons vu que ces méthodes permettent de calculer la stratégie optimale de deux NSP face à un client si les deux NSP collaborent. L'utilisation de méthodes de la Théorie des Jeux nous permet de déduire un équilibre de Nash et une stratégie optimale égoïste dans le cas où les deux NSP ne collaborent pas, et ainsi déduire le Prix de l'Anarchie d'une telle situation. Les résultats concernant deux NSP face à un client peuvent très facilement se généraliser au cas de n NSP.

Cependant, l'application de ces méthodes à un cas réel de négociation de SLA pose de nombreux problèmes :

- **Paramètres inconnus du NSP :** De nombreux paramètres utilisés pour calculer une stratégie optimale d'un NSP ne sont pas connus par celui-ci dans un cas de figure réel. La fonction d'échec f peut grossièrement être inférée statistiquement, mais n'est jamais connue avec précision dans la réalité (de par son caractère stochastique et la complexité du réseau). La fonction d'utilité du client est également inconnue par les NSP, ils ne peuvent pas savoir à priori quel SLA maximisera l'utilité du client, ni anticiper la décision du client face à plusieurs offres. De même, les offres des NSP concurrents ne sont pas connues.
- **La scalabilité :** Les méthodes utilisées sont polynomiales en fonction du nombre d'états du système. Pour un NSP i, le nombre d'états dans le pire des cas est de $(|\mathcal{Q}_i|+1)^\Delta$, où $|\mathcal{Q}_i|$ est le nombre de SLA dont dispose le NSP i et Δ est la durée d'un SLA en unités de temps. Si la granularité temporelle est assez fine, le nombre d'états devient trop grand pour pouvoir utiliser ces méthodes.
- **La dynamicité :** L'ensemble des SLA dont disposent les NSP, la durée des SLA, les requêtes des clients, et bien d'autres paramètres sont dynamiques et évoluent au cours du temps. Les stratégies optimales devraient donc être recalculées à chaque changement, ce qui serait prohibitif en terme de temps d'exécution et de coût.

De plus, le modèle devient beaucoup plus complexe s'il intègre plusieurs clients. Ces problèmes nous poussent à utiliser des méthodes plus flexibles, robustes à la dynamicité, qui passent à l'échelle et qui ne nécessitent pas la connaissance à priori de tout le modèle (probabilités d'échecs, choix du client, etc.) pour fournir de bonnes solutions.

2.7 Algorithmes d'apprentissage

Les algorithmes d'apprentissage réunissent tous ces avantages. Ils ne nécessitent pas de connaître tous les paramètres du modèle, leur scalabilité peut être paramétrée indépendamment de la taille du modèle, et ils peuvent réapprendre après une modification du système. C'est pourquoi nous avons choisi de les adapter au problème de la négociation de SLA et d'évaluer leur efficacité. Dans cette section, nous présenterons d'abord le principe général des algorithmes d'apprentissage, puis nous décrirons les trois algorithmes que nous avons choisis et leurs caractéristiques (paramètres, convergence, etc.). L'adaptation de ces algorithmes au problème de négociation de SLA sera présentée dans les sections suivantes.

2.7.1 Principe général des algorithmes d'apprentissage

Le principe des algorithmes d'apprentissage est d'effectuer des actions, d'observer l'effet de ces actions et la réaction de son environnement, et enfin de déduire des informations permettant de *mieux*

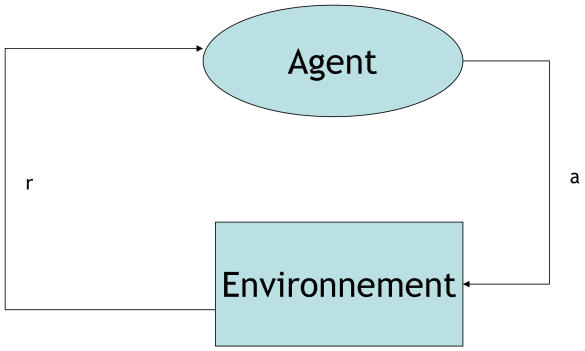


FIGURE 2.5 – Schéma fonctionnel de l'interaction entre un agent et son environnement.

choisir (selon une certaine mesure) les futures actions. Le schéma en figure 2.5 explicite ce principe de fonctionnement. Au début, l'agent qui apprend ne sait rien. Il choisit une action (au hasard ou selon une politique donnée pré-implémentée) à effectuer, puis observe la réaction de l'environnement. La réaction de l'environnement est valuée par l'agent, c'est-à-dire qu'il a un ordre de préférence sur les différentes réactions possibles et peut leur associer une valeur (un gain, par exemple). L'observation de cette réaction permet à l'agent de déduire des informations et d'adapter sa politique de choix d'action selon ce qu'il a appris. Le but de l'agent est d'effectuer les actions qui lui sont favorables, par exemple de maximiser son gain. Cette boucle est répétée un certain nombre de fois jusqu'à atteindre une condition d'arrêt. L'agent considère son environnement comme une boîte noire à laquelle il soumet des actions en entrée et de laquelle il récupère des informations (valeur, gain, etc.) en sortie. Il n'a donc pas besoin de connaître le fonctionnement interne de cette boîte et déduira des informations uniquement des couples (entrée/sortie).

Même si le but de l'agent est de maximiser son gain, il a deux manières antagonistes d'y parvenir :

- **L'exploration :** Cela consiste à choisir toutes les actions un grand nombre de fois pour affiner l'estimation de l'agent sur le gain que rapporte chacune des actions. Plus une action est choisie, plus l'estimation du gain qu'elle rapporte est précise.
- **L'exploitation :** Cela consiste à choisir le plus souvent les actions qui rapportent le plus grand gain (selon les information actuelles de l'agent). L'avantage est qu'on ne prend pas de risque à essayer des actions qui s'avèrent rapporter un gain insuffisant (ou nul), mais on diminue la probabilité de trouver de nouvelles actions qui rapportent un gain encore supérieur à celui qu'on attend.

L'usage habituel est de privilégier l'exploration en début d'apprentissage puis de favoriser de plus en plus l'exploitation quand l'estimation de l'agent se fait plus précise.

Dans la section suivante, nous allons décrire les trois algorithmes que nous avons choisi d'adapter. Le premier est choisi pour sa très grande simplicité et son efficacité dans la plupart des cas. Les deux suivants sont plus complexes mais, entre autres avantages, sont *model free*, c'est-à-dire qu'ils ne nécessitent pas une connaissance du modèle sur lequel ils apprennent.

2.7.2 L'algorithme Linear Reward Inaction (LRI)

L'algorithme Linear Reward Inaction [17] (LRI) est un algorithme d'apprentissage très ancien qui a d'abord été décrit dans le domaine de la psychologie mathématique [28] avant d'être popularisé en tant que méthode d'apprentissage automatique [105].

Dans sa version initiale, le LRI fonctionne sur un système de récompense binaire ; soit l'action réussit et l'agent reçoit une récompense, soit l'action échoue et l'agent ne reçoit rien. L'idée de l'algorithme est de maintenir un vecteur de probabilités p_t : une probabilité $p_t[i]$ pour chaque action a_i dans l'ensemble des actions $\mathcal{A}$. Si une action réussit, on augmente la probabilité lui correspondant ; si une action échoue, on ne modifie pas les probabilités. La mise à jour des probabilités est telle que la somme des probabilités dans le vecteur vaut toujours 1. L'action suivante est choisie en fonction de la distribution de probabilités du vecteur. L'algorithme est le suivant :

17. *Linear Reward Inaction :* Linéaire - Récompense - Inaction.

Algorithme 1 L'algorithme LRI

$t \leftarrow 0$
Initialiser $p_t[j] \leftarrow \frac{1}{|\mathcal{A}|}$ pour chaque $a_j \in \mathcal{A}$
loop
  Sélectionner une action a_i en fonction de la distribution de probabilités $p_t[.]$
  Si l'action a_i réussit Alors
    $p_{t+1}[i] \leftarrow p_t[i] + b(1 - p_t[i])$
    $p_{t+1}[j] \leftarrow p_t[j](1 - b)$ pour tout $j \neq i$
  Sinon
    $p_{t+1}[j] \leftarrow p_t[j]$ pour toutes les actions $a_j \in \mathcal{A}$
  $t \leftarrow t + 1$
end loop

Le *taux d'apprentissage*, noté b, est un paramètre permettant de régler la vitesse d'apprentissage et de convergence. Il est tel que $0 < b \leq 1$. Le LRI converge toujours vers une configuration où la probabilité d'une action est à 1 et celle de toutes les autres est à 0 [63]. Néanmoins, l'action avec une probabilité de sélection à 1 n'est pas toujours une action qui réussit. Le LRI peut converger vers une action qui échoue toujours, mais plus b est petit, plus les chances de converger vers une telle action sont faibles.

Le LRI peut être adapté à un contexte autre que la récompense binaire. En effet, si on suppose que les actions rapportent des gains r_t tels que $0 \leq r_t \leq 1$. On peut modifier la formule de mise à jour des probabilités de la manière suivante :

$$p_{t+1}[i] \leftarrow p_t[i] + b \cdot r_t(1 - p_t[i]) \text{ où } a_i \text{ est l'action choisie}$$

$$p_{t+1}[j] \leftarrow p_t[j](1 - b \cdot r_t) \text{ pour tout } j \neq i$$

Si le gain rapporté par les actions n'est pas majoré par 1, il suffit de remplacer r_t par $\frac{r_t}{r_{\max}}$ dans les formules ci-dessus (où $r_{\max}$ est la plus grande valeur possible de gain associée à une action).

2.7.3 Les algorithmes de Reinforcement Learning : Q-Learning et SARSA

Les algorithmes de Reinforcement Learning [18] ont le même principe que les autres algorithmes d'apprentissage, sauf qu'ils n'apprennent pas quelle action choisir dans l'absolu mais selon les situations. En fait, l'agent sera associé à un ensemble d'états et il apprendra quelle action choisir dans tel ou tel état. Les algorithmes de Reinforcement Learning sont définis sur un modèle markovien que nous décrivons dans la section qui suit.

2.7.3.1 Markov Decision Process (MDP)

Un Markov Decision Process [19] (MDP) est un quadruplet $\langle S,A,R(.,.),P(.,.,.)\rangle$ où :
- S est un ensemble fini d'états,
- $A = \bigcup_{s \in S} A_s$ est un ensemble fini d'actions représentant les décisions que peut prendre l'agent. L'ensemble A_s est l'ensemble des actions que l'agent peut effectuer quand il est dans l'état s,
- $R(s,a)$, $s \in S$, $a \in A$ est la fonction de revenu (ou de gain) représentant le gain de l'agent quand il choisit l'action a en étant dans l'état s,
- $P(s,a,s')$ est la fonction de probabilité conditionnelle de transition. C'est la probabilité d'atteindre l'état s' après avoir choisi l'action a en étant dans l'état s.

Une action est choisie à chaque *instant de décision* (*Decision epoch* en anglais), qui est une unité de temps discrète (qui correspond à l'unité de temps utilisée dans le modèle). Si le nombre d'instants de décision est fini, on dit que le MDP est à horizon fini, sinon il est à horizon infini. Une politique Π est une distribution de probabilités sur A_s pour chaque $s \in S$. Etant dans un état s, $\Pi(a)$ est la probabilité d'effectuer l'action a. S'il existe pour chaque état une action a telle que $\Pi(a) = 1$, la politique Π est dite déterministe, autrement elle est dite stochastique.

18. *Reinforcement Learning :* apprentissage par renforcement
19. *Markov Decision Process :* Processus de Décision Markovien

L'objectif de l'agent est de maximiser son *revenu total actualisé* (*Discounted total reward* en anglais) à partir d'un instant de décision t, noté $\mathcal{R}_t$, qui est défini par :

$$\mathcal{R}_t = \sum_{t=0}^{T} \gamma^t R_t(s,a) \cdot P(s,a,s') \tag{2.8}$$

où $R_t(s,a)$ est le gain obtenu à l'instant t, l'horizon $T \in \mathbb{N} \cup \{+\infty\}$ et γ, tel que $0 < \gamma < 1$, est le facteur d'actualisation. Cette somme représente la somme des gains sur une période (finie ou infinie), mais où les gains sont pondérés. Plus t est loin dans le futur, plus le poids qui est associé à son gain est faible.

Pour plus de détails sur les MDP, le lecteur pourra consulter l'ouvrage de référence de Puterman [94].

2.7.3.2 Q-valeur d'un couple (état, action)

Les algorithmes de Reinforcement Learning que nous allons présenter apprennent la *Q-valeur* de tous les couples (état, action). La Q-valeur d'un couple (état, action), notée $Q(s,a)$, est définie comme suit :

$$Q(s,a) = \mathbb{E}\left[\mathcal{R}_t | s_t = s, a_t = a\right]$$

où s_t est l'état actuel (à l'instant t), et a_t est l'action choisie à l'instant t. La Q-valeur optimale d'un couple (état, action) est le revenu total actualisé maximum associé à ce couple.

2.7.3.3 L'algorithme du Q-Learning

L'algorithme du Q-Learning at été proposé par Watkins en 1992 [115]. Comme son nom l'indique, cet algorithme *apprend* les Q-valeurs optimales de chaque couple (état, action). Une fois qu'il aura convergé vers ces valeurs, il peut en déduire une politique optimale de choix des actions. Généralement, l'algorithme termine après une nombre d'itération fixé a priori par l'utilisateur.

Algorithme 2 Algorithme du Q-Learning

```
Initialisation des Q-valeurs (à 0 où une valeur quelconque uniforme)
loop
  A chaque instant de décision t faire
    Choisir une action a_t selon une politique quelconque basée sur les Q-valeurs
    Observer le gain r_t et le nouvel état s_{t+1}
    Mettre à jour les Q-valeurs selon la formule (2.9)
end loop
```

L'algorithme 2 correspond au Q-Learning, son principe est de choisir une action selon une politique basée sur les Q-valeurs, d'observer le gain obtenu et le nouvel état dans lequel il est, puis de mettre à jour les Q-valeurs selon la formule suivante :

$$Q_{t+1}(s,a) = (1-\alpha_t)Q_t(s,a) + \alpha_t(r_t + \gamma \max_{a' \in A_{s'}} Q_t(s',a')) \tag{2.9}$$

où r_t est le gain observé à l'instant t, s_{t+1} est le nouvel état observé, γ est le facteur d'actualisation et α_t est un taux d'apprentissage qui évolue au cours du temps. La manière dont α_t est mis à jour impacte particulièrement la convergence de l'algorithme.

2.7.3.4 SARSA

L'algorithme SARSA (State-Reward-Action-State-Reward) proposé par Rummery et Niranjan [100] fonctionne selon le même principe. La seule différence est dans la formule de mise à jour des Q-valeurs qui est la suivante :

$$Q_{t+1}(s,a) = (1-\alpha_t)Q_t(s,a) + \alpha_t(r_t + \gamma Q_t(s',a'))$$

Ici, l'action a' utilisée pour la mise à jour n'est pas l'action qui maximise le revenu dans l'état s', mais l'action réelle choisie une fois dans l'état s'. Cette action est choisie selon une politique basée sur les Q-valeurs, tout comme l'action a.

2.7.3.5 Convergence

L'algorithme du Q-Learning converge vers les Q-valeurs optimales sous deux conditions [72] :

1. Tous les couples (état, action) doivent être visités une infinité de fois,
2. $\sum_{t=0}^{\infty} \alpha_t = \infty$ et $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$.

Les critères de convergence ont été améliorés par Even-Dar et Mansour [41] : si la mise à jour de α_t est polynomiale, i.e., $\alpha_t = \frac{1}{(t+1)^\omega}$, avec $\frac{1}{2} < \omega < 1$, alors le temps de convergence est polynomial en fonction de $\frac{1}{1-\gamma}$. Cependant, si $\omega = 1$, le temps de convergence est exponentiel en fonction de $\frac{1}{1-\gamma}$.

Les conditions de convergence de SARSA sont légèrement différentes [108] :

- Tous les couples (état, action) doivent être visités une infinité de fois,
- La politique de choix de l'action doit être déterministe quand $t \to +\infty$. C'est-à-dire que la probabilité de choisir l'action qui maximise la Q-valeur $Q(s,a)$ tend vers 1 quand t tend vers l'infini.

2.7.3.6 Les politiques de choix basées sur les Q-valeurs

Une politique de choix basée sur les Q-valeurs est la manière dont on choisit l'action à effectuer dans un état s en fonction des valeurs $Q(s,a)$ pour les différentes actions a. Cette politique doit permettre l'exploration de l'environnement, c'est-à-dire essayer plusieurs couples (état, action) pour apprendre les Q-valeurs, mais doit également permettre l'exploitation, c'est-à-dire de choisir plus souvent les actions qui ont une Q-valeurs élevée pour maximiser les gains. Initialement, l'agent doit apprendre et donc privilégier l'exploration. Mais après convergence, l'exploitation doit avoir la priorité.

Il existe plusieurs politiques possibles, les plus utilisées sont :

Politique déterministe gloutonne : Cette politique consiste à sélectionner l'action qui maximise la Q-valeur : $a_t = \underset{a \in A_s}{\text{argmax}}\ Q_t(s,a)$.

politique ϵ-gloutonne : Cette politique sélectionne l'action ayant la Q-valeurs la plus élevée avec une probabilité $1-\epsilon$, et une action aléatoire avec une probabilité ϵ. Le paramètre ϵ est initialisé à une valeur élevée afin d'encourager l'exploration, puis elle décroit quand les Q-valeurs sont plus précises. Cette politique converge vers une politique déterministe et gloutonne quand $\epsilon \to 0$:

$$P(a_t = a) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A_s|} & \text{si } a = \text{argmax } Q_t(s,a) \\ \frac{\epsilon}{|A_s|} & \text{sinon} \end{cases}$$

La politique *softmax* : Cette politique sélectionne une action avec une probabilité proportionnelle à sa Q-valeur. Cette politique n'est pas forcément gloutonne quand $t \to \infty$:

$$P(a_t = a') = \frac{Q_t(s,a')}{\sum_{a \in A_s} Q_t(s,a)}$$

La politique de Boltzmann : Cette politique choisit une action en fonction de l'exponentielle de sa Q-valeur. Un paramètre τ permet de contrôler la différence entre les probabilités des différentes actions. Quand $\tau \to \infty$, l'écart entre les probabilités tend vers 0 et la politique devient aléatoire uniforme. Quand $\tau \to 0$, l'écart entre les probabilités est maximisé et la politique tend vers la politique déterministe gloutonne. Habituellement, τ est initialisé à une grande valeur pour encourager l'exploration, puis il décroit pour privilégier l'exploitation :

$$P(a_t = a') = \frac{e^{Q_t(s,a')/\tau}}{\sum_{a \in A_s} e^{Q_t(s,a)/\tau}} \quad \tau \in \mathbb{R}_+^*$$

		NSP 2		
		q_2^1	q_2^2	q_2^3
NSP 1	q_1^1	1/2, 1/2	0, 1	1, 0
	q_1^2	1, 0	1/2, 1/2	0, 1
	q_1^3	0, 1	1, 0	1/2, 1/2

TABLE 2.1 – Gains des NSP en fonction du SLA qu'ils proposent.

2.8 Comparaisons des différentes politiques : simulations

2.8.1 Jeu sans ordre de préférence sur les actions

Dans le modèle de prix et d'utilité que nous avons présenté en section 2.3, le client a un ordre de préférence total sur les SLA que lui proposent les NSP. Ce n'est pas toujours le cas. Il se peut que la préférence du client ne soit pas transitive, c'est-à-dire qu'un client préfère un SLA q_i^1 à q_i^2, q_i^2 à q_i^3 et q_i^3 à q_i^1. Si les deux NSP proposent le même SLA, le client choisira l'un d'eux aléatoirement. On peut modéliser cette situation sous forme d'un jeu très connu : *Pierre, Papier, Ciseaux* (ou *Chifoumi*). Le tableau 2.1 montre l'espérance de gain pour chaque NSP en fonction du SLA offert par lui-même et son adversaire dans une telle situation.

2.8.2 Paramètres de simulation

Le jeu présenté en section 2.8.1 a été simulé durant 10000 rounds, où un round est une répétition de 100 étapes élémentaires (une étape consiste en une requête du client, une offre de chaque NSP puis le choix du client et la rémunération du NSP). Pour chaque round, le résultat affiché est la moyenne des 100 étapes qui le composent.

Le LRI a été implémenté dans deux versions : une version où le taux d'apprentissage b est fixe, et une autre où il décroit graduellement. Pour la première version, la valeur choisie est $b = 0{,}01$. Pour la deuxième version, la valeur initiale est également $b = 0{,}01$ puis décroit en fonction du temps pour tendre vers 0.

Concernant le Q-Learning utilisant la politique ϵ-gloutonne, le paramètre ϵ est initialisé à 0,01 puis décroit en fonction du temps. Si le Q-Learning utilise la politique de Boltzmann, le paramètre τ est initialisé à 0,9. Dans tous les cas, le paramètre d'apprentissage du Q-Learning est initialisé à $\alpha_0 = 0{,}6$ puis décroit polynomialement en fonction du temps comme recommandé dans [41].

Les paramètres de SARSA ont été fixés aux mêmes valeurs que les paramètres du Q-Learning.

2.8.3 Résultats de simulation contre un concurrent qui n'apprend pas

Pour donner un aperçu de la convergence des algorithmes d'apprentissage avec différentes politiques, nous avons effectué une simulation où le NSP 1 (dont on observe les choix) utilise différentes stratégies ou algorithmes et le NSP 2 (le concurrent) offre toujours le SLA q_2^3. Evidemment, le NSP 1 devra apprendre que l'offre à faire est le SLA q_1^1, car il est sur d'être choisi par le client.

La figure 2.6 montre la fréquence à laquelle le SLA q_1^1 est offert par le NSP 1. Très clairement, les deux versions du LRI ont un comportement identique et convergent extrêmement rapidement. Au bout de quelques dizaines de rounds, le LRI (dans ses deux versions) n'offre plus que le SLA qui le fera sélectionner par le client. Etrangement, le Q-Learning et SARSA, utilisant tous deux la politique de Boltzmann, commencent par converger très rapidement puis se bloquent à un certain seuil de fréquence. Les autres politiques donnent à peu près les mêmes résultats avec SARSA et le Q-Learning et convergent lentement.

2.8.4 Résultats de simulation contre un concurrent qui apprend

La figure 2.7 montre les revenus par round du NSP observé (NSP 1) quand celui-ci utilise différent algorithmes d'apprentissage contre un adversaire (NSP 2) qui apprend également. Les paramètres des

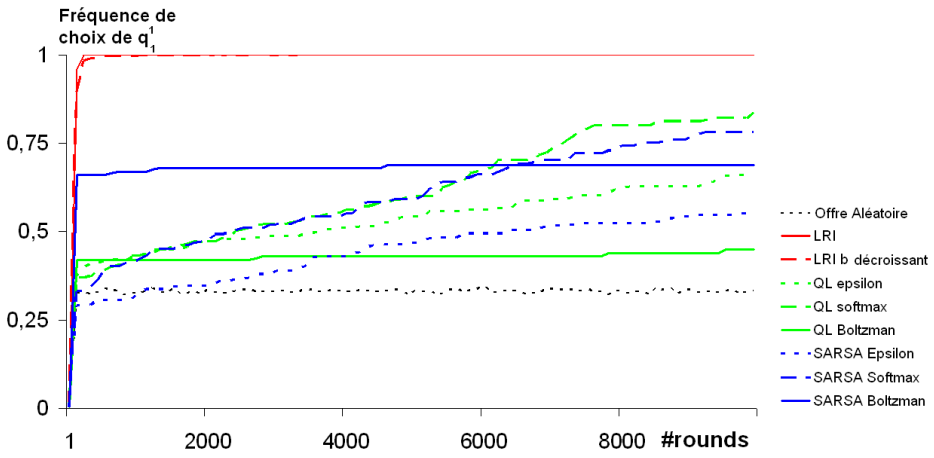


FIGURE 2.6 – Fréquence d'offre du SLA q_1^1 lorsque le NSP 2 propose systématiquement le SLA q_2^3.

différents algorithmes sont ceux donnés en section 2.8.2.

La figure 2.7(a) montre l'évolution du revenu par round du NSP 1 quand celui-ci utilise le LRI et que son adversaire utilise différents algorithmes. Les revenus les plus élevés sont obtenus contre le Q-Learning et SARSA avec une politique ϵ-gloutonne. Le revenus contre le LRI (dans ses deux versions), le Q-Learning et SARSA avec politique de Boltzmann sont quasiment identiques autour de 20% du revenu maximum. Enfin, les revenus contre un adversaire qui utilise la politique *softmax* (que ce soit avec le Q-Learning ou SARSA) commencent à un niveau assez élevé mais décroissent continuellement en approchant de 0%. Il semble donc que le LRI se comporte le mieux contre la politique ϵ-gloutonne en maintenant des revenus entre 30 et 40% du revenu maximum. Il se comporte mal contre le LRI et la politique de Boltzmann. Enfin, bien que la convergence rapide du LRI lui assure des revenus élevés au début contre la politique *softmax*, celle-ci devient meilleure au fil du temps et le LRI finit par ne plus gagner du tout. Il semblerait donc que, sur le long terme, la politique *softmax* soit la meilleure à opposer au LRI.

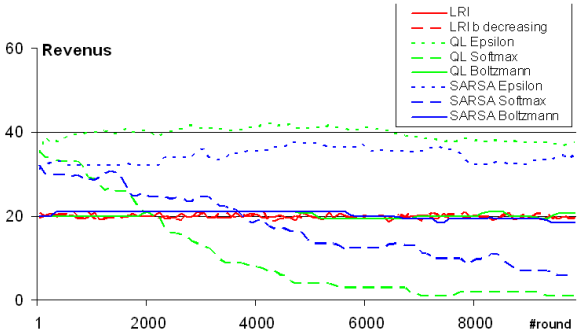
Les résultat obtenu quand le NSP 1 utilisent le LRI avec b décroissant sont presque identiques à ceux obtenus avec le LRI classique. Nous avons donc choisi de ne pas les présenter car ils n'apportent aucune nouvelle information.

La figure 2.7(b) montre l'évolution du revenu du NSP 1 quand lui-même et son adversaire utilisent le Q-Learning avec différentes politiques. Quand les deux NSP utilisent la politique ϵ-gloutonne ou Boltzmann, les résultats sont assez bons (autour de 60% pour la politique de Boltzmann et 50% pour la politique ϵ-gloutonne. La politique *softmax* se comporte très mal contre la politique ϵ-gloutonne, les revenus tendant vers 0%. La politique de Boltzmann ne se comporte pas très bien face aux deux autre politiques.

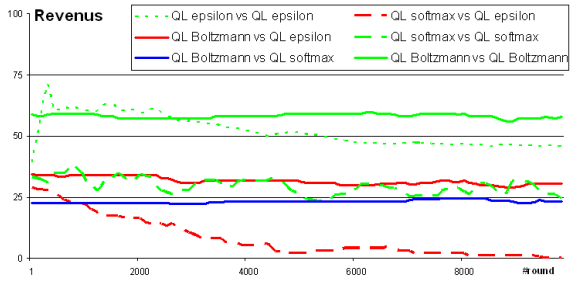
La figure 2.7(c) montre l'évolution du revenu du NSP 1 quand les deux NSP utilisent l'algorithme SARSA avec différentes politiques. La politique de Boltzmann se comporte très bien face à la politique *softmax* avec environs 73% du revenu maximum. Les revenus sont autour de 50% quand les deux NSP utilisent la politique de Boltzmann. Enfin, la politique *softmax* finit par ne plus gagner du tout face à la politique ϵ-gloutonne.

La figure 2.7(d) montre l'évolution du revenu du NSP 1 quand celui-ci utilisent SARSA et que le NSP 2 utilisent le Q-Learning. Les revenus sont les plus élevés sont obtenus par SARSA avec la politique de Boltzmann contre le Q-Learning utilisant la politique ϵ-gloutonne ou *softmax* (plus de 60% du revenu maximum). Les revenus les plus bas sont obtenus quand le NSP 1 utilise SARSA avec la politique *softmax*.

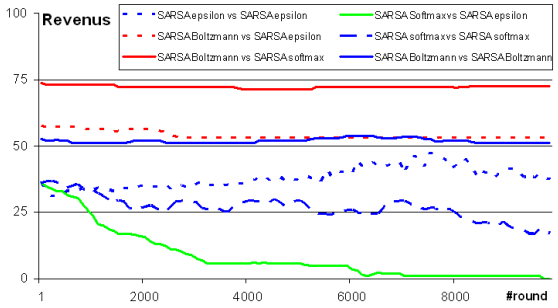
Les résultats de ces simulations ne montrent pas un ordre total des algorithmes et des politiques d'apprentissage selon l'efficacité. Il semble néanmoins que la politique d'apprentissage ait une plus grande influence que l'algorithme en lui-même. La politique de Boltzmann donne de bons résultats contre la politique *softmax*, qui elle-même donne de bons résultats contre le LRI sur le long terme. Néanmoins, le LRI converge très rapidement et est à privilégier sur le court terme.



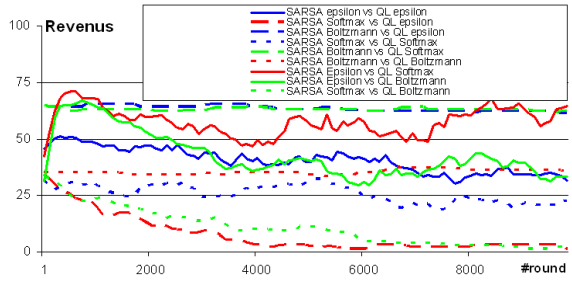
(a) Revenus en utilisant le LRI contre un adversaire qui apprend



(b) Revenus en utilisant le Q-Learning contre un adversaire qui utilise le Q-Learning



(c) Revenus en utilisant SARSA contre un adversaire qui utilise SARSA



(d) Revenus en utilisant le SARSA contre un adversaire qui utilise le Q-Learning

FIGURE 2.7 – Revenus d'un NSP quand l'autre apprend également.

2.9 Conclusion

La négociation de SLA est une étape centrale dans l'établissement de routes en inter-domaine avec QoS. Le problème crucial d'un NSP et savoir quel SLA proposer pour maximiser ses gains à long terme. Trouver une stratégie permettant d'offrir ce SLA n'est pas trivial car celle-ci va dépendre de beaucoup de paramètres : fonction de choix du client, prix des SLA, capacité du NSP, présence ou non de concurrence, etc.

Dans un premier temps, nous nous sommes intéressés au problème quand le NSP fournisseur est en situation de monopole et que le client accepte forcément l'offre qui lui est faite. Nous avons modélisé le choix de l'offre comme étant un problème d'optimisation et avons proposé une solution exacte basée sur les algèbres $(\max, +)$. Les mêmes outils nous ont permis de déterminer la stratégie optimale en présence de deux NSP qui collaborent. Si ceux-ci ne collaborent pas et sont égoïstes, leur stratégie peut différer et le gain obtenu peut être moindre que celui obtenu quand ils collaborent. Nous avons montré que le Prix de l'Anarchie peut être arbitrairement grand. Les résultats obtenus avec deux NSP (en collaboration ou en concurrence) se généralisent très facilement à n. C'est par soucis de simplicité que nous nous sommes contentés de présenter le cas avec deux NSP.

Néanmoins, les méthodes utilisées pour la résolution exacte du problème nécessitent une connaissance parfaite de tous les paramètres (choix du client, offre des autres NSP, capacité, etc.), paramètres que le NSP ne peut pas connaître totalement. L'utilisation de méthode approchées indépendante des paramètres est donc nécessaire. Certains algorithmes d'apprentissage nous ont paru adaptés car ils sont *model free*, c'est-à-dire qu'ils ne nécessitent pas de connaître le modèle ni ses paramètres pour apprendre une solution. Nous avons donc adaptés certains de ces algorithmes (LRI, Q-Learning, SARSA) au problème de négociation de SLA et avons observé leur comportement dans une situation où le client n'a pas d'ordre de préférence strict sur les SLA. Le LRI s'est montré le plus rapide en ce qui concerne la convergence mais n'est pas forcément à privilégier sur le long terme. Aucun algorithme ne s'est révélé être le meilleur dans l'absolu. Le choix d'un algorithme plutôt qu'un autre dépend de la situation.

3 Négociation de SLA avec mécanisme de réputation

3.1 Introduction

Tous les résultats établis dans le chapitre précédent sont basés sur l'hypothèse que les clients ne sont sensibles qu'aux paramètres de QoS et au prix des SLA. Néanmoins, le plus souvent, les clients sont sensibles à un troisième paramètre : la réputation du fournisseur.

Nous considérerons la réputation d'un NSP comme une valeur exprimant sa fiabilité, c'est-à-dire la proportion de SLA qui n'échouent pas. Dès lors, la question que nous nous posons est la suivante : étant donné un tel mécanisme de réputation, comment un NSP peut prendre cette réputation en compte pour calculer sa stratégie optimale ? Sachant que la réputation évolue au cours du temps, la stratégie optimale peut également évoluer selon les valeurs que prend la réputation. La méthode exacte employée dans le chapitre précédent pour analyser la compétition entre les NSP ne peut plus s'appliquer si on considère une réputation qui évolue dans le temps. En effet, la valeur de la réputation va influencer le choix du client, les probabilités de transition entre les états seront dépendantes de la réputation, donc du temps, et la chaîne de Markov modélisant le système ne sera plus homogène.

Le problème est complexe car le NSP doit faire face à un dilemme. Comme nous l'avons vu dans le chapitre précédent, la probabilité de d'échec (ou de violation) d'un SLA dépend du taux de capacité utilisée du NSP. Or si celui-ci provisionne trop de SLA pour maximiser son gain immédiat, son taux de capacité utilisée augmentera considérablement, ce qui augmentera la probabilité d'échec des SLA, et fera donc baisser la réputation du NSP. Or, une baisse de la réputation du NSP diminuera ses chances d'être sélectionné par un client, ce qui fera baisser l'espérance de gain à long terme. D'un autre coté, si le NSP veut garder une bonne réputation pour maximiser ses gains à long terme, il n'a pas d'autre choix que de provisionner peu de SLA pour maintenir un taux de capacité utilisée bas, et donc une probabilité de violation de SLA basse. Paradoxalement, cette stratégie maintiendra des gains peu élevés puisque le nombre de SLA provisionnés sera bas. On peut constater que provisionner trop de bande passante induira des revenus bas, mais provisionner trop peu de SLA aura la même conséquence. Le problème est donc de calculer la stratégie qui donnera le compromis optimal entre une capacité utilisée trop basse (peu de gains) et une capacité utilisée trop élevée (baisse de la réputation, et donc des gains à long terme).

Dans ce chapitre, nous présenterons d'abord un modèle de réputation simple qui exprime la fiabilité des NSP. N'ayant pas de solution analytique pour calculer la stratégie optimale dans ce contexte, nous nous proposerons plutôt d'adapter les algorithmes d'apprentissage vus dans le chapitre précédent au problème de la négociation de SLA avec mécanisme de réputation. Car si le NSP ne peut pas calculer analytiquement sa stratégie optimale, il peut toujours l'apprendre. Nous verrons comment adapter le modèle de MDP pour qu'il prenne en compte la valeur de la réputation. Enfin, nous comparerons par simulation ces algorithmes avec des stratégies simples de sélection de SLA et montrerons la supériorité des méthodes d'apprentissage sur les stratégies simples.

Ce chapitre est basé sur les articles [C1] et [W1] publiés dans le cadre de cette thèse.

3.2 Mécanisme de réputation dans les réseaux

Dans les années 2000, l'étude de la réputation dans les réseaux s'est principalement focalisée sur les mécanismes de réputation et de notation sur les sites de vente en ligne ([35, 98], etc.) et sur les réseaux sociaux ([101, 57, 73], etc.). Dans ce contexte, la réputation est définie comme une valeur que les membres d'un réseau (utilisateurs, acheteurs, etc.) attribuent à l'un d'entre eux (utilisateur, vendeur, etc.).

La formalisation théorique de la réputation diffère grandement selon le domaine et les travaux. Dans leur ouvrage de référence, Mailath et Samuelson [74] définissent la réputation comme étant un certain équilibre dans un jeu répété. En fait ils considèrent la réputation comme étant une valeur binaire, soit *haute*, soit *basse*. Garder une réputation haute revient à rester dans un certain équilibre, perdre sa réputation haute (et donc avoir une réputation basse) revient à dévier de cet équilibre. L'ensemble des valeurs que peut prendre la réputation peut être étendu à un ensemble fini où chaque valeur correspond à un certain équilibre. Les transitions entre les valeurs dépendent des actions des différents acteurs. Cette approche a notamment été utilisée pour concevoir des protocoles incitant les membres de sites de services collaboratifs à participer aux tâches et à effectuer les services en question [121]. D'autres travaux [32] considèrent la réputation comme étant une préférence pour un fournisseur plutôt qu'un autre et la modélisent par un bruit augmentant la probabilité de choisir le fournisseur préféré. Cette notion de réputation ne nous intéresse pas car elle n'évolue pas dans le temps et ne dépend pas des actions des acteurs.

L'application d'un mécanisme de réputation à la négociation de SLA a été proposée dans quelques travaux [96, 36]. Ceux-ci proposent une classification des violations de SLA (partielles, totales, pondérées) dans un mécanisme de pénalités et discutent de la relation entre le monitoring et un éventuel système de réputation. Ces travaux proposent des mécanismes généraux et abstraits mais n'explicitent aucune méthode pour calculer la réputation ni la manière dont elle doit impacter la négociation de SLA. Sans évoquer la notion de réputation, d'autres travaux [13, 14] proposent d'estimer la fiabilité (probabilité de non violation de SLA) dans les systèmes de gestion des grilles de calcul (appelés GRID). Ils ne proposent pas de mécanisme de calcul explicite mais listent plusieurs possibilités et scénarios. L'interprétation de la réputation en tant que probabilité (de succès) semble l'approche la plus intéressante pour la négociation de SLA, car cette information est la plus déterminante pour le client.

Nous supposons que tous les clients ont connaissance de la réputation de tous les NSP. Nous ne nous intéressons pas au mécanisme de propagation de la réputation, ni à la manière dont les clients partagent cette information. Le lecteur intéressé peut se référer aux chapitres 23 et 27 de [87] pour une vue d'ensemble des systèmes de réputation et des mécanismes de propagation.

3.3 Modèle avec réputation

Comme dit précédemment, nous avons opté pour un modèle simple de réputation qui exprime la fiabilité du NSP, autrement dit la probabilité (empirique) qu'un SLA provisionné par un NSP donné n'échoue pas. Nous essayons autant que possible de garder le même modèle que dans le chapitre précédent pour la négociation de SLA (section 2.3). Les deux points suivants sont les seuls qui en diffèrent :

- La formule définissant la réputation d'un NSP est ajoutée au modèle,
- Étant donnée l'hypothèse que le client est sensible à la réputation, son utilité (et sa fonction de choix) est modifiée afin de prendre en compte ce paramètre.

3.3.1 Réputation d'un NSP

Nous considérons la réputation d'un NSP i à l'instant t comme étant le ratio entre le nombre de SLA provisionnés par le NSP i qui n'ont pas échoué et le nombre total de SLA provisionnés par ce même NSP. Cette valeur peut-être vue comme étant simplement la probabilité (empirique) de succès des SLA du NSP i :

$$\mathrm{rep}_t(i) = 1 - \frac{\#fail(i)}{\#select(i)}$$

où $\#fail(i)$ est le nombre de SLA provisionnés par le NSP i entre l'instant 0 et l'instant t et qui ont échoué, et $\#select(i)$ et le nombre total de SLA provisionnés entre 0 et t. Nous rappelons que seuls les offres acceptées par un client sont provisionnées.

3.3.2 Utilité d'un client

Rappelons tout d'abord la fonction d'utilité d'un client par rapport à un SLA, comme définie par l'équation 2.1 :

$$U(\mathrm{q}_i^j) = ||\mathrm{q}_i^j|| - \eta \frac{p_i^j}{p_{\max}}$$

où q_i^j est le SLA offert par le NSP i, la fonction $||\cdot||$ est la mesure de QoS, p_i^j est le prix du SLA q_i^j, la borne $p_{\max}$ est le prix maximum que le client est prêt à payer. Enfin, η est un paramètre propre à chaque client qui pondère le prix par rapport à la QoS.

La réputation d'un NSP peut être vue comme la probabilité de succès des SLA offerts par ce NSP. Nous considérons donc que cette valeur pondère l'utilité du client par rapport à un SLA. Et nous définissons la nouvelle utilité qui prend en compte la réputation :

$$U^{\mathrm{rep}}(\mathrm{q}_i^j) = \mathrm{rep}_t(i) \cdot U(\mathrm{q}_i^j) \tag{3.1}$$

Notons simplement q_i le SLA offert par le NSP i, et numérotons les NSP de 1 à m. Le choix du client se fera aléatoirement en fonction de la nouvelle utilité du client. La probabilité qu'une offre soit choisie est égale à l'utilité par rapport à cette offre divisée par la somme des utilités par rapport à toutes les offres :

$$\Pr[\text{Sélectionner } \mathrm{q}_i] = \frac{U^{\mathrm{rep}}(\mathrm{q}_i)}{\sum_{k=1}^{m} U^{\mathrm{rep}}(\mathrm{q}_k)}$$

Cette valeur peut être différente selon les clients vu que certains paramètres de la fonction d'utilité diffèrent (notamment le paramètre η) d'un client à un autre.

3.3.3 Problématique

Vu que le choix du client dépend de la valeur de la réputation, et que celle-ci évolue au court du temps, si on modélise le système en tant que chaîne de Markov, les probabilités de transition entre les états sont dépendantes du temps et la chaîne de Markov n'est pas homogène. Pour éviter les difficultés analytiques inhérentes à l'étude d'un tel système, nous nous proposons d'adapter les méthodes d'apprentissage vues au chapitre précédent au problème de négociation de SLA avec mécanisme de réputation.

Le but d'un NSP sera de maximiser ses gains en apprenant une stratégie optimale ou proche de l'optimale. Si la granularité des SLA est assez fine, trouver une telle stratégie est équivalent à déterminer le valeur optimale du taux de capacité utilisée. En effet, si le taux de capacité utilisée s'approche de 0, cela signifie que le NSP ne provisionne aucun SLA et que ses gains sont nuls. Si le taux de capacité utilisée s'approche de 100%, la probabilité d'échec des SLA tend vers 1 et les gains seront nuls (rappelons que dans notre modèle, si un SLA échoue le NSP rembourse le prix du SLA au client).

Une solution analytique paraît trop complexe à trouver. Mais ce n'est pas la seule motivation pour l'utilisation d'algorithmes d'apprentissage. Même sans réputation, une solution analytique suppose la connaissance totale de la fonction d'utilité et de choix du client, ce qui n'est généralement pas le cas de la part des NSP. Une telle solution suppose également la connaissance de la fonction d'échec qui donne pour chaque valeur de taux de capacité utilisée la probabilité d'échec correspondante. Cette fonction peut être grossièrement approximée empiriquement, mais il est impossible de la connaître précisément car elle dépend de beaucoup de paramètres : la topologie du réseau du NSP, la loi d'arrivée et le type de trafic, les politiques de routage, de protection, etc. L'avantage des algorithmes d'apprentissage que nous utilisons est qu'ils sont *model-free*[20], c'est-à-dire qu'ils peuvent apprendre et converger sans connaissance préalable des probabilités de transition entre les états, et ces probabilités dépendent en partie de la fonction d'échec. Ces algorithmes peuvent donc apprendre sans aucune connaissance préalable de la fonction d'échec.

3.4 Capacité utilisée, probabilité d'échec et états du NSP

Les algorithmes d'apprentissage doivent être adaptés au mécanisme de réputation. Les résultats du chapitre précédent nous ont montré que le Q-Learning et SARSA ne présentent que de faibles différences de résultats, nous nous contenterons d'évaluer le LRI et le Q-Learning lors de la prise en compte de la

20. Indépendant du modèle

Niveau de capacité	Taux de capacité utilisée	Probabilité d'échec de SLA
0	$\rho = 1$	1
1	$0{,}95 \leq \rho < 1$	0,99
2	$0{,}90 \leq \rho < 0{,}95$	0,90
3	$0{,}80 \leq \rho < 0{,}90$	0,80
4	$0{,}60 \leq \rho < 0{,}80$	0,50
5	$0{,}30 \leq \rho < 0{,}60$	0,20
6	$\rho < 0{,}30$	0,05

TABLE 3.1 – Relation entre niveau de capacité, taux de capacité utilisée et probabilité de violation de SLA.

réputation. Concernant le LRI, aucune modification n'est nécessaire car c'est un algorithme *sans états* (i.e., la probabilité de choisir une action est indépendante de l'état du NSP). Cependant, le Q-Learning se base sur les états du MDP pour effectuer l'apprentissage. Ces états doivent donc prendre en compte la capacité restante du NSP ainsi que les requêtes qu'il reçoit.

3.4.1 Discrétisation de la capacité utilisée

Le taux de capacité utilisée ρ doit être dans les états du MDP pour que le Q-Learning puisse apprendre des stratégies qui prennent en compte la capacité restante du NSP. Cependant, la création d'un état pour chaque valeur de $\rho \in [0,1]$ impliquerait un nombre infini (et même non dénombrable) d'états. Pour éviter ce problème, le taux de capacité utilisée doit être discrétisé. Il est converti en *niveau de capacité*, noté G. Le tableau 3.1 montre un exemple de discrétisation. Comme la probabilité d'échec d'un SLA est proportionnelle au taux de capacité utilisée, celle-ci est également discrétisée pour qu'à chaque niveau de capacité G corresponde une probabilité d'échec déterminée. Le tableau 3.1 montre le niveau de capacité correspondant à chaque intervalle de valeurs de ρ, ainsi que la probabilité d'échec qui lui est associée.

3.4.2 Redéfinition du MDP

Un état s du MDP est un couple (req,G) où req est un profile de requêtes (un ensemble de paramètres de QoS que le client demande) et G est le niveau de capacité actuel du NSP. S'il n'y a qu'un seul client, le profil de requête n'a pas d'importance car nous supposons qu'un client déterminé envoie toujours le même profile de requêtes. Cependant, s'il y a plusieurs clients, il se peut que chacun d'eux envoie un profile de requêtes différent. Mettre le profile de requêtes dans les états du MDP permet donc au NSP d'adapter sa stratégie en fonction des différents clients.

Pour le reste, les paramètres du MDP sont comme suit :

- S est l'ensemble des états du MDP comme défini ci-dessus. Chaque couple possible (req,G) est un état. Il y a donc $|\text{req}| \times |G|$ états, où $|\text{req}|$ est le nombre total de types de requêtes et $|G|$ est le nombre de niveaux de capacité du NSP. Définir un état par niveau de capacité plutôt que par la liste de SLA qui sont provisionnés permet davantage de scalabilité. En effet, le nombre de niveaux de capacité est fixé indépendamment du nombre de SLA et de leur durée, il ne dépend que de la précision et de la granularité souhaitées.
- A est l'ensemble des actions. Il correspond simplement à l'ensemble des SLA dont dispose le NSP plus l'action de ne proposer aucun SLA (ne pas faire d'offre),
- $P(s,j,s')$ est la probabilité d'atteindre l'état s' en étant dans l'état s et en offrant le SLA q_i^j,
- $R(s,j,s')$ est le gain perçu par le NSP en atteignant l'état s' après à partir de l'état s et après avoir offert le SLA q_i^j. Ce gain correspond au prix du SLA si celui-ci est accepté par le client, et à 0 s'il n'est pas accepté.

La figure 3.1 montre une partie d'un MDP associé à un NSP. On peut y voir l'état actuel s_t et les différents états futurs possibles s_{t+1}. Les probabilités de transitions dépendent de la probabilité d'échec du SLA choisi, des offres des autres NSP et du choix du client.

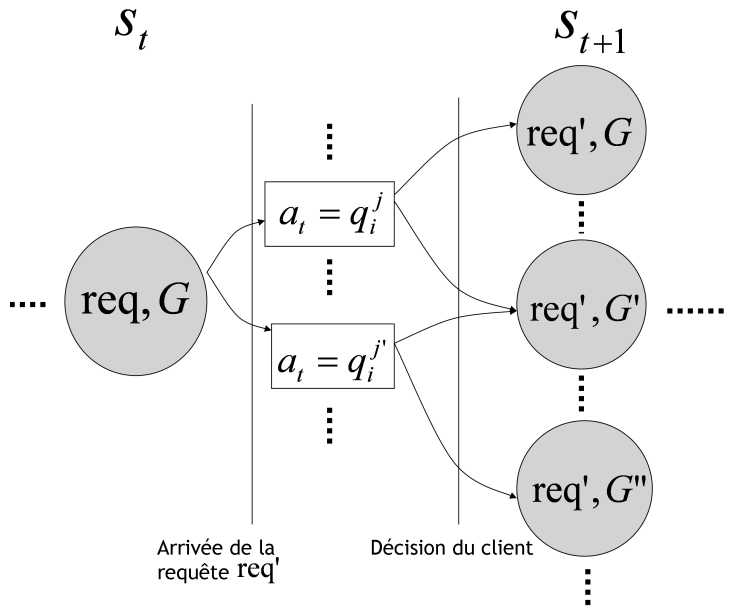


FIGURE 3.1 – Une partie du MDP d'un NSP où l'on voit les différentes transitions possibles entre l'état à l'instant t et l'état à l'instant $t+1$.

3.5 Résultats de simulation

Afin d'évaluer l'efficacité des algorithmes d'apprentissage (Q-Learning et LRI) et leur capacité à apprendre des stratégies prenant en compte la réputation du NSP, nous avons effectué des simulations sur deux topologies :

1. Un client envoyant des requêtes à deux NSP simultanément.
2. Deux clients envoyant chacun des requêtes à deux NSP simultanément.

3.5.1 Algorithmes simulés

Anfin d'évaluer leur efficacité, le LRI et le Q-Learning sont comparés à deux stratégies triviales :
- **Stratégie Min :** Cette stratégie consiste à offrir le SLA le moins cher qui satisfait la requête du client.
- **Stratégie Uniforme :** Cette stratégie consiste à offrir aléatoirement et avec des probabilités uniformes un des SLA qui satisfait la requête du client.

3.5.2 Simulations avec un client et deux NSP

3.5.2.1 Paramètres de simulations

Nous présentons les résultats de simulations effectuées sur la topologie présentée en figure 3.2. Chaque NSP dispose de quatre SLA, chaque SLA est décrit dans le tableau 3.2. Les SLA sont présentés dans un ordre croissant de prix.

SLA	Bande passante	délai	Taux de perte
q_i^1	100Mbps	20ms	0.1%
q_i^1	200Mbps	20ms	0.1%
q_i^2	250Mbps	30ms	0.1%
q_i^3	2500Mbps	5ms	0.001%

TABLE 3.2 – L'ensemble de SLA du NSP i ($i = 1,2$).

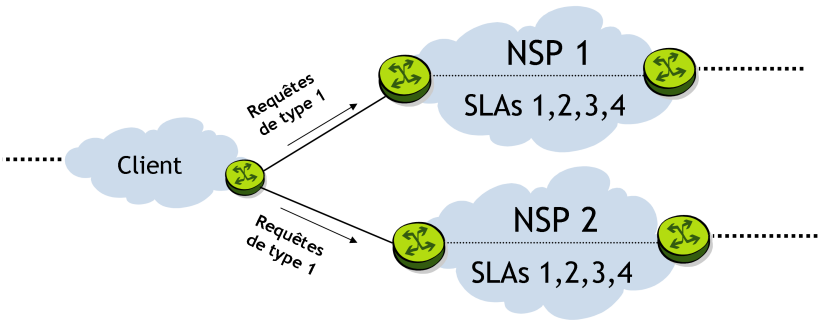


FIGURE 3.2 – Topologie avec un client connecté à deux NSP.

Nous avons fixé la durée des SLA à $\Delta = 40$ unités de temps et la capacité maximale des NSP à 10000Mbps chacun. Les requêtes du client ont les mêmes paramètres que le SLA q_i^1, sauf le délai qui est de 30ms. Tous les SLA satisfont donc les requêtes du client. Concernant les paramètres d'apprentissage, nous avons fixé le taux d'apprentissage du LRI, $b = 0{,}02$ et le taux d'apprentissage du Q-Learning, $\alpha_0 = 0{,}5$. Le facteur d'actualisation du Q-Learning est fixé à $\gamma = 0{,}8$.

Les simulations ont été effectuées durant 10000 rounds, chaque round a été répété 100 fois. Les résultats affichés sont les moyennes sur les 100 répétitions de chaque round.

Remarque : Nous employons indifféremment les termes « algorithme » et « stratégie » dans les commentaires des résultats de simulations. Chaque NSP utilise une stratégie particulière. Chaque courbe est notée « Algorithme 1 vs. Algorithme 2 », où « Algorithme 1 » est la stratégie utilisée par le NSP dont on observe les résultats, et « Algorithme 2 » est la stratégie utilisée par le NSP concurrent. *QL* signifie « Q-Learning », *Min* signifie « Stratégie Min », *Uniform* signifie « Stratégie Uniforme ». Quand les deux NSP utilisent le même algorithme, la courbe est notée « Algorithme » au lieu de « Algorithme 1 vs. Algorithme 2 ».

3.5.2.2 Résultats

La figure 3.3 montrent le gain moyen **par requête** du NSP 1 utilisant différentes stratégies. Les résultats montrent qu'après une courte phase d'apprentissage, le gain moyen se stabilise, sauf celui obtenu avec le LRI qui continue à croitre lentement. Le NSP 1 obtient les gains les plus élevés quand il utilise le LRI et que son concurrent utilise la Stratégie Uniforme. Les meilleurs gains suivants sont obtenus quand le NSP 1 utilise le Q-Learning et que son concurrent utilise la Stratégie Uniforme. Comme on peut s'y attendre, la Stratégie Min obtient les gains les moins élevés contre toutes les autres stratégies, ce qui est logique car elle propose systématiquement le NSP le moins cher. On peut constater que le LRI obtient de meilleurs résultats que le Q-Learning, et que généralement les algorithmes d'apprentissage se comportent bien mieux que les stratégies triviales.

La figure 3.4 montre l'évolution de la réputation du NSP 1 au cours du temps. Evidemment, la Stratégie Min permet d'avoir la réputation la plus élevée contre toutes les autres stratégies. Vu que la Stratégie Min choisit toujours le SLA le moins cher (qui est celui qui a les paramètres de QoS les plus bas), cette stratégie minimise la capacité utilisée du NSP 1 et donc minimise aussi la probabilité de violation des SLA, ce qui permet de garder une réputation élevée. Toutes les autres stratégies stabilisent la réputation entre 0,8 et 0,9, ce qui semble être le meilleur compromis permettant d'avoir des gains élevés tout en gardant une bonne réputation. Le LRI induit une réputation plus basse que le Q-Learning.

La figure 3.5 montre l'évolution de la capacité utilisée par le NSP 1. Plus la capacité utilisée est élevée, plus les gains sont importants. Cependant, toutes les stratégies maintiennent la capacité utilisée entre 20% et 45% (sauf la Stratégie Min qui stagne sous les 20%).

3.5.3 Simulations avec deux clients et deux NSP

Cette section présente les résultats de simulations effectués sur la topologie présentée en figure 3.6. Tous les paramètres sont les mêmes que dans le cas d'un seul client. La seule différence est la présence d'un deuxième client qui envoie les mêmes requêtes que le premier client.

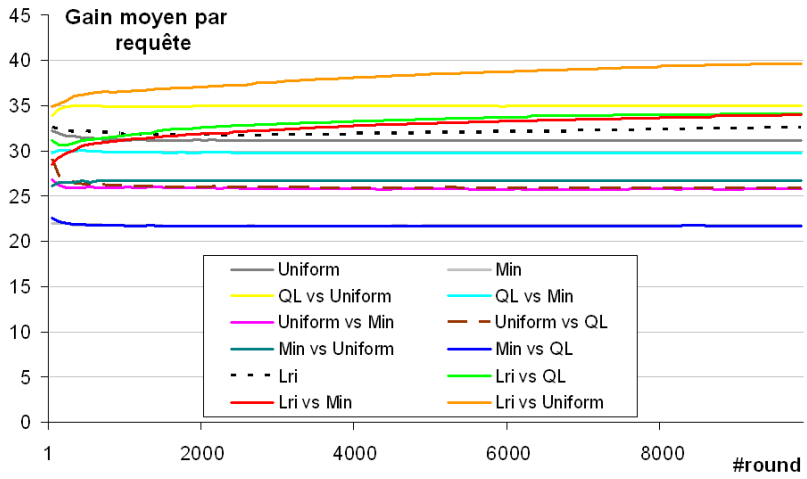


FIGURE 3.3 – Gains par requête du NSP 1.

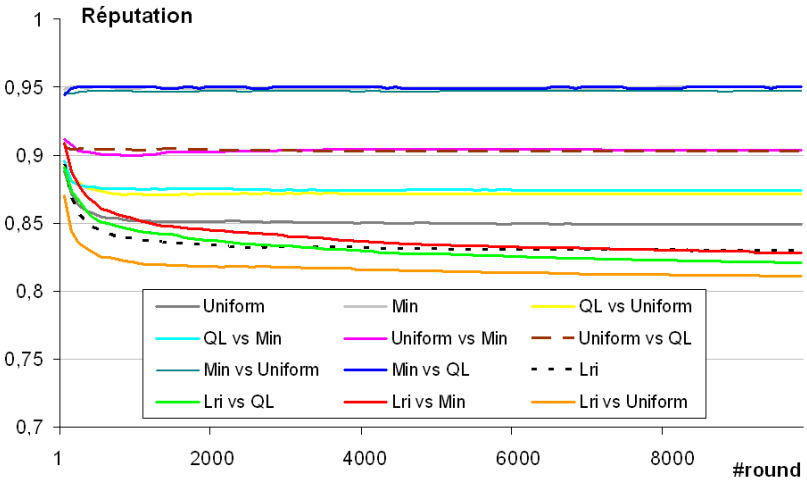


FIGURE 3.4 – Evolution de la réputation du NSP 1.

La figure 3.7 montre l'évolution du gain moyen **par requête** du NSP 1. Il obtient le gain le plus élevé en utilisant le LRI, quelque soit la stratégie du concurrent. Il semble que le LRI soit largement meilleur que le Q-Learning face à l'arrivée de requêtes simultanées des deux clients. Le Q-Learning montre des résultats moins bons que dans le cas d'un seul client. ici encore, la Stratégie Min obtient les moins bons résultats.

Comme on peut le voir sur la figure 3.8, l'évolution de la réputation suit la même tendance que dans le cas d'un seul client. La valeur de la réputation est inversement proportionnelle aux gains moyens obtenus par le NSP. La réputation est plus basse que dans le cas d'un seul NSP, ce qui s'explique évidemment par le fait que les NSP utilisent plus de capacité pour satisfaire les deux clients. Par exemple, la réputation du NSP 1 utilisant le LRI contre un concurrent utilisant la Stratégie Uniforme se stabilise autour de 0,81 quand il y a un seul client et autour de 0,67 quand il y a deux clients.

La figure 3.9 montre l'évolution de la capacité utilisée par le NSP 1 selon différentes stratégies. Comme dans le cas d'un seul client, la capacité utilisée est proportionnelle au gain et inversement proportionnelle à la réputation du NSP. La capacité utilisée en présence de deux clients n'atteint pas le double de la capacité utilisée en présence d'un seul client. En effet, la capacité utilisée (de même que le gain) se comporte de manière sous-additive. Si on double le nombre de requêtes, la capacité utilisée augmente mais ne double pas, car le taux de violation de SLA augmente, ce qui fait baisser le gain moyen par requête.

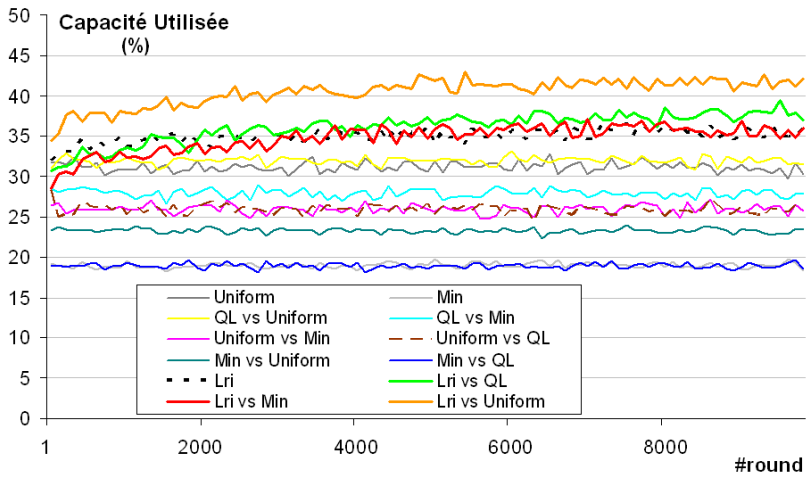


FIGURE 3.5 – Capacité utilisée du NSP 1 (%).

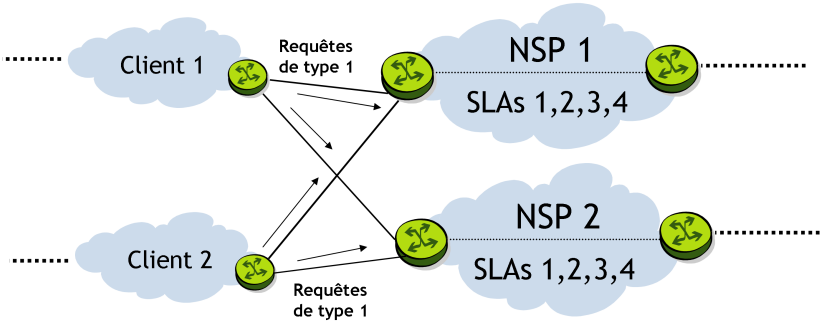


FIGURE 3.6 – Topologie avec deux clients connectés à deux NSP.

3.5.4 Interprétation des résultats

Dans le cas d'un seul client, la première constatation est que les algorithmes d'apprentissage donnent de bien meilleurs résultats que les stratégies triviales Min et Uniforme. Le gain moyen par requête peut varier du simple au double entre la Stratégie Min et le LRI. Il est évident que les algorithmes d'apprentissage ont déduit quels SLA avaient une espérance de gain élevée face au stratégies triviales. Quand les deux NSP utilisent des algorithmes d'apprentissage, les résultats sont moins bons que face aux stratégies triviales. Le fait que le concurrent apprenne en même temps perturbe l'apprentissage.

Le LRI rapporte plus de gains que le Q-Learning face à n'importe quelle stratégie. Ce constat semble surprenant étant donné que le Q-Learning dispose d'un modèle plus riche distinguant les différents états du NSP, ce qui aurait dû aboutir à un apprentissage plus fin selon les états. Néanmoins, la prise en compte des états présente un inconvénient : l'apprentissage effectué dans un état ne sert à rien dans les autres états. Le Q-Learning doit effectuer un apprentissage distinct dans chaque état, ce qui ralentit considérablement la convergence. Le LRI converge beaucoup plus rapidement que le Q-Learning, et une fois sa convergence effectuée, il propose le SLA qui maximise son espérance de gain (et qui maximise la probabilité d'acceptation du client). La vitesse de convergence élevée du LRI est l'explication la plus vraisemblable de sa meilleure performance. Il semble également que la prise en compte des états du NSP ne soit pas déterminante : après convergence, les NSP sont en « *régime de croisière* » et restent dans le même état, l'apprentissage dans les autres états n'aura pas été utile car ils ne seront que très rarement visités. La finesse d'apprentissage qu'apporte la prise en compte des états ne contrebalance pas l'effet négatif de ralentissement de convergence qu'elle induit.

En présence de deux clients, le LRI surpasse clairement le Q-Learning, qui n'est que légèrement meilleur que la Stratégie Uniforme. Là encore, il semble que la vitesse de convergence a joué un rôle déterminant. Néanmoins, même si le LRI est meilleur que le Q-Learning dans toutes les configurations,

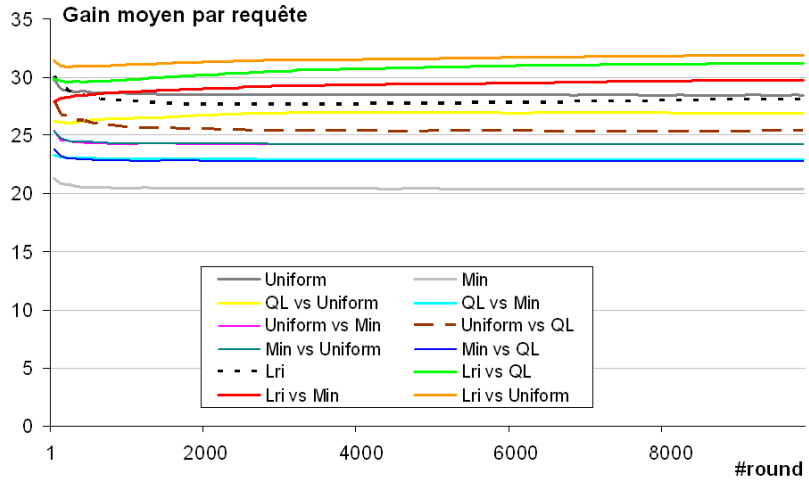


FIGURE 3.7 – Le gain moyen du NSP 1 par requête.

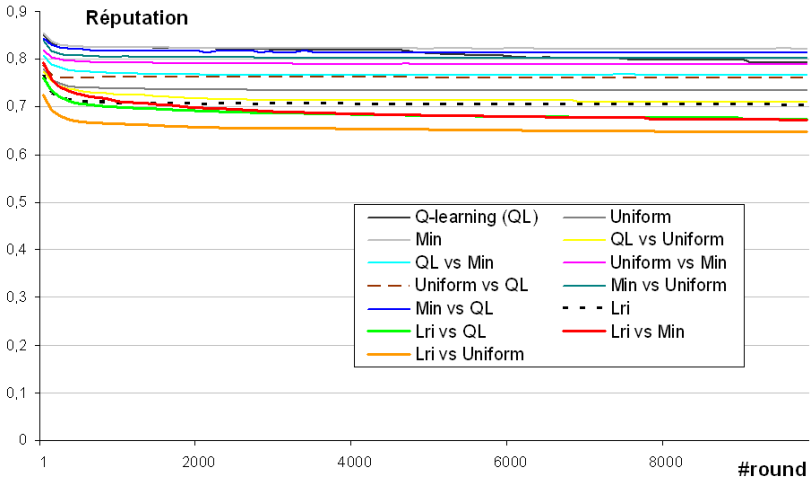


FIGURE 3.8 – L'évolution de la réputation du NSP 1.

il ne faut pas oublier que les performances des algorithmes d'apprentissage dépendent également de la stratégie du NSP concurrent. On peut voir que le Q-Learning est bien meilleur face à la Stratégie Uniforme que face à la Stratégie Min. Il n'est donc pas exclu qu'il existe certaines stratégies particulières contre lesquelles le Q-Learning pourrait être meilleur que le LRI.

3.6 Conclusion

Dans la négociation de SLA, il est plausible que les clients soient sensibles à la QoS des SLA et leur prix, mais également à la réputation des NSP fournisseurs, considérant que la réputation est une valeur exprimant la fiabilité d'un NSP. Dès lors, le problème de calcul d'une stratégie optimale pour un NSP devient beaucoup plus complexe. En effet, il doit trouver un compromis entre une réputation trop basse (qui hypothèque ses gains futurs) et une capacité utilisée trop basse (qui se traduit par des gains peu élevés). Trouver ce compromis est d'autant plus difficile que beaucoup de paramètres cruciaux qui influent sur la stratégie optimale ne sont pas connus du NSP : la fonction d'utilité du client et ses paramètres, la probabilité d'échec des SLA, etc.

L'avantage de l'utilisation des algorithmes d'apprentissage est qu'ils ne nécessitent pas de connaître explicitement ces paramètres. Nous avons donc appliqué ces algorithmes au problème de négociation de SLA avec mécanisme de réputation. L'application du Q-Learning a nécessité la modification des états

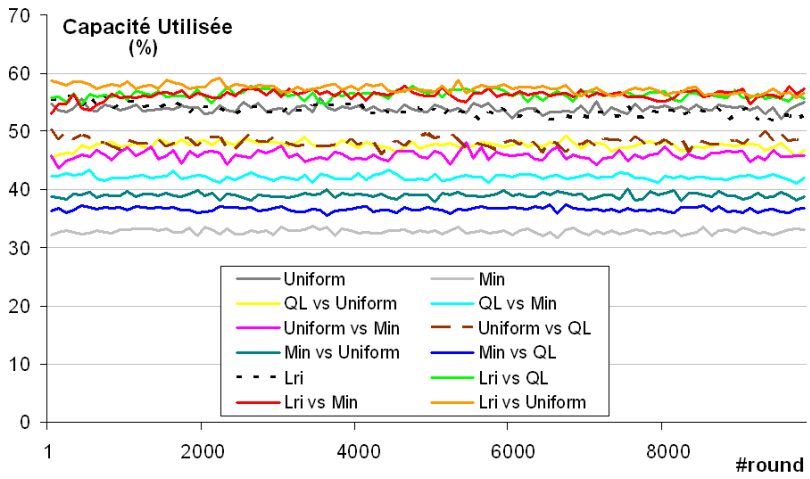


FIGURE 3.9 – La capacité utilisée du NSP 1 en présence de deux clients.

du MDP, qui prennent en compte le niveau de capacité utilisée par le NSP ainsi que le profile de requête qu'il reçoit. Ces paramètres ne dépendent ni du nombre de SLA dont dispose le NSP ni de leur durée. Cette définition des états du NSP permet donc une bien meilleure scalabilité que la définition vue au chapitre précédent.

Lors des simulations effectuées avec un ou deux clients et deux NSP, nous avons pu constater que les algorithmes d'apprentissage donnent de bien meilleurs résultats que des stratégies triviales consistant à offrir le SLA le moins couteux ou offrir un SLA aléatoirement. De manière surprenante, le LRI s'est montré bien plus efficace que le Q-Learning, qui dispose pourtant d'un modèle plus riche prenant en compte les différents états du NSP. La cause la plus probable est la rapidité de convergence du LRI.

Deuxième partie

Calcul de chemins dans les réseaux hétérogènes

4

Hétérogénéité, tunnels et encapsulations dans les réseaux

4.1 Introduction

Après la négociation de SLA entre les domaines, un chemin *abstrait* (consistant en une liste de domaines) est établi. Ce chemin doit être composé d'une liste de SLA mis bout à bout. Néanmoins, les SLA ne sont pas instanciés avant la négociation et le processus de négociation est agnostique aux technologies utilisées (pour laisser les domaines choisir librement). Le chemin (liste de liens internes au domaine) qu'ils vont emprunter dans chaque domaine n'est pas forcément préétabli, ainsi que plusieurs autres caractéristiques techniques : les protocoles à utiliser, les technologies mises en œuvre, etc. Pour que le chemin de bout en bout soit instancié, il faut déterminer précisément les liens qui seront utilisés dans les domaines et entre les différents domaines (liens d'interconnexion) ainsi que les technologies déployées et les protocoles utilisés.

Or, calculer un chemin traversant différents domaines n'est pas trivial puisque se pose le problème de l'incompatibilité des technologies et/ou protocoles. En effet, les domaines utilisent souvent différentes technologies et différents protocoles qui ne sont pas toujours compatibles. Actuellement, la convergence sur IP permet la communication entre les différents domaines sur Internet. Néanmoins, plusieurs autres protocoles coexistent, certains à l'intérieur d'un même domaine (le cœur d'un domaine n'utilise pas forcément le protocole IP pour acheminer les données), d'autres entre les domaines (des liens d'interconnexion non IP). Ces liens et ces protocoles doivent être pris en compte dans l'exploration (ou le calcul) de chemins pour augmenter le nombre de solutions possibles.

L'architecture Pseudo-Wire [26] définit des fonctions d'encapsulations et de désencapsulations (appelées *fonctions d'adaptation*) pour pallier ce problème. Certains protocoles sont encapsulés dans d'autres pour traverser des portions du réseau qui utilisent des technologies différentes. Tous les nœuds du réseau ne disposent évidemment pas de ces fonctions. Pour calculer un chemin, il faut donc prendre en compte ces fonctions et leur emplacement dans le réseau. Le problème est complexe et sera traité en détail dans le chapitre suivant.

Dans ce chapitre, nous nous proposons de présenter les problèmes d'hétérogénéité entre les domaines et comment le concept d'encapsulation permet de les prendre en compte dans les réseaux d'aujourd'hui. La section 4.2 explique brièvement et de façon abstraite les notions d'encapsulation et de désencapsulation ainsi que l'impact qu'elle ont sur le calcul de chemins. Les encapsulations sont liées à l'organisation en couches des réseaux, la section 4.3 décrit donc le modèle historique OSI qui structure les fonctionnalités des réseaux en 7 couches et présente un exemple classique d'encapsulation. Néanmoins, les encapsulations les plus souvent utilisées ne respectent pas forcément ce modèle. L'architecture Pseudo-Wire, qui est réellement déployée aujourd'hui, définit de nombreuses encapsulations de protocoles en dehors du modèle OSI. L'architecture Pseudo-Wire et certaines des ses encapsulations sont présentées en section 4.4. Il est possible de mettre bout à bout plusieurs segments de Pseudo-Wire, l'architecture qui le permet, appelée Pseudo-Wire multisegment, est présentée dans la même section ainsi que la problématique de calcul de chemins dans une telle architecture. Par souci de généralité, d'autres architectures où la même problématique de calcul de chemins se pose sont présentées en section 4.5. Enfin, la section 4.6 conclut le chapitre.

4.2 Encapsulation, désencapsulation et chemins faisables

Cette section décrit brièvement et de manière informelle les notions d'encapsulation, de désencapsulation et de faisabilité d'un chemin. Des exemples réels techniques d'encapsulations et de désencapsulations seront présentés plus loin dans ce chapitre.

4.2.1 Unité de données de protocole

Les informations transmise selon un protocole donnée sont structurées en unités appelées *Unités de Données de Protocoles* (Protocol Data Units, PDU). Les PDU de différents protocoles sont différents mais ont une méta-structure commune (voir la figure 4.1) :

1. **Un en-tête :** Il contient les données nécessaires et utiles à l'acheminement du PDU, par exemple : identité (ou adresse) du destinataire, identifiant du PDU, etc.
2. **Un champ de données :** Il contient les données utiles, c'est-à-dire les informations que l'on veut réellement transmettre.
3. **Une extension ou remplissage :** Il n'est présent que dans le cas où le protocole définit une taille minimum de PDU. Ce champ est rempli par une valeur quelconque (soit aléatoire, soit fixée par convention) de façon à ce que le PDU atteigne la taille minimum nécessaire. Il ne contient aucune information utile.

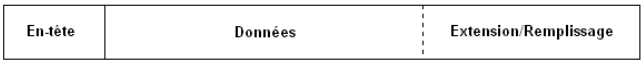


FIGURE 4.1 – Structure d'une unité de données de protocole.

4.2.2 Encapsulation et désencapsulation de protocoles

On dit qu'un PDU d'un protocole A est encapsulé dans un PDU d'un protocole B (et par extension que le protocle A est encapsulé dans le protocole B) quand le PDU entier de A (en-tête et données) est placé dans le champ *données* du PDU de B. La figure 4.2 montre le résultat d'une telle encapsulation.

FIGURE 4.2 – Un PDU du protocole A encapsulé dans un PDU du protocole B.

La désencapsulation est le processus de récupération du PDU encapsulé. L'en-tête et l'éventuel remplissage du PDU de protocole B sont supprimés et le PDU de A est récupéré. Les encapsulations et désencapsulations se font au niveau de certains nœuds du réseau.

4.2.3 Calcul de chemin avec prise en compte des encapsulations

Actuellement, le calcul de chemins dans un réseau disposant d'encapsulations et de désencapsulations se fait manuellement. La première étape d'une automatisation serait de disposer d'un algorithme de calcul de chemins prenant en compte les contraintes de compatibilité entre les protocoles.

Considérons le réseau comme un ensemble de nœuds et de liens (possiblement orientés) entre eux. Certains nœuds ont la capacité d'encapsuler certains protocoles dans d'autres, certains nœuds (pas forcément les mêmes) peuvent désencapsuler certains protocoles. Le calcul de chemins doit prendre en compte les encapsulations et désencapsulations de protocoles ainsi que leur possible imbrication. En s'abstrayant des protocoles et technologies spécifiques, une solution générique devra non seulement déterminer l'ensemble des nœuds d'un chemin, mais également les protocoles faisant partie du chemin. Le chemin doit être *faisable* dans le sens où à chaque encapsulation doit correspondre une désencapsulation plus loin dans le chemin. Par exemple, si un nœud encapsule des PDU de protocole A dans des PDU de

protocole B, les PDU de A doivent être extraits des PDU de B plus tard. Ces contraintes sont appelées *contraintes de compatibilité de protocoles* et elles doivent être transparentes aux entités communicantes aux deux extrémités du chemin.

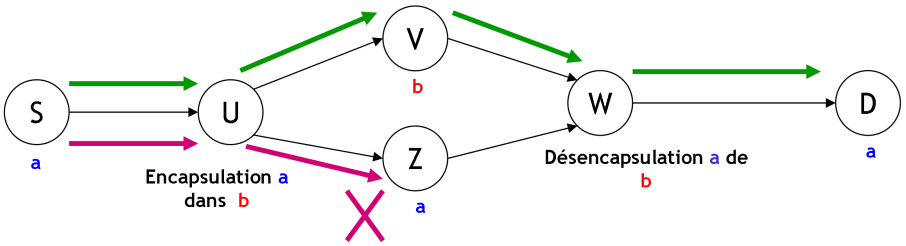


FIGURE 4.3 – Exemple de réseau avec un chemin faisable (en vert) et un autre non faisable (en rouge).

La figure 4.3 montre un exemple abstrait de réseau. Les fonctions d'encapsulation et désencapsulation sont spécifiées en-dessous de chaque nœud. Par exemple, le nœud U est capable d'encapsuler des PDU de protocole A dans des PDU de protocole B. Le nœud V peut uniquement transmettre les PDU de B tels qu'il les reçoit. Ici, le chemin S,U,V,W,D en vert est faisable car le nœud U reçoit des PDU de A et les encapsule dans des PDU de B avant de les transmettre au nœud V. Le nœud V transmettra les PDU de B au nœud W qui va désencapsuler (ou extraire) les PDU de A des PDU de B avant de les envoyer au nœud D. Le chemin S,U,Z,W,D en violet n'est pas faisable car le noeud Z reçoit des PDU de A contenant des PDU de B, alors qu'il ne peut que traiter des PDU de A.

Les algorithmes de calcul de chemins classiques (Dijkstra, Bellman-Ford, etc.) ne peuvent pas résoudre ce problème car ils ne prennent pas en compte les contraintes de compatibilité. Le problème peut-être résolu par une recherche exhaustive, mais sa complexité serait prohibitive. On sait que le problème de trouver un chemin faisable est NP-complet si on y ajoute une contrainte de bande passante [67], mais aucune solution polynomiale au problème sans bande passante n'était connue. Nous détaillerons ce problème et proposerons des solutions dans le chapitre suivant.

Outre le problème de décision (l'existence d'un chemin faisable), plusieurs problèmes d'optimisation se posent :

1. **Minimiser le nombre de sauts :** Un objectif classique serait, si un chemin faisable existe, de trouver celui qui comprend le minimum de sauts (et donc de nœuds traversés). En effet, le passage par un nœud (qui représente un routeur) peut induire un délai considérable correspondant au temps de séjour du paquet dans les files d'attente du nœud. Le délai est ainsi fortement corrélé au nombre de sauts. Passer par moins de nœuds permet aussi de réduire la charge moyenne des nœuds.
2. **Minimiser le nombre d'encapsulations :** L'encapsulation peut occasionner des délais importants dans le chemin (surtout dus à la fragmentation, au réordonnancement et au regroupement de paquets comme nous le verrons dans l'architecture Pseudo-Wire en section 4.4.1.). Si ces délais sont largement supérieurs à ceux induits par le passage passif (sans encapsulation) par un nœud, il serait intéressant de minimiser non pas le nombre de sauts mais le nombre d'encapsulations (et donc de désencapsulations) dans le chemin. Cela réduirait également le nombre de nœuds à configurer.
3. **Minimiser la longueur du chemin :** En attribuant un poids (ou un coût) à chaque lien et/ou à chaque fonction d'adaptation, il serait utile de pouvoir trouver le chemin faisable qui minimise la somme de ces poids. Ce serait une généralisation du point 1 à n'importe quelle métrique additive. Cette métrique pourrait être le coût en énergie, le délai, la distance, etc. Cela peut être également utile dans le cas où l'ont doit calculer un chemin sous plusieurs contraintes de QoS. En effet, plusieurs heuristiques proposent de combiner les contraintes de QoS et de les exprimer en une seule valeur [82]. Une fois cette combinaison faite pour chaque lien, calculer le plus court chemin selon cette nouvelle métrique peut servir de solution de départ à une heuristique.

Dans le chapitre suivant, nous proposerons la première solution polynomiale à ces problèmes.

4.3 Modèle en couche des réseaux

Cette section présente la structuration *en couches* des réseaux. Cette structuration est à l'origine de la présence d'encapsulations et de désencapsulations dans le réseau.

4.3.1 Le modèle OSI

Afin de gérer la complexité des réseaux et disposer d'un standard dans la conception de services réseaux, l'*International Organization for Standardization* (ISO) a proposé le modèle *Open Systems Interconnection* (OSI) [61]. Le modèle OSI est une décomposition en couches fonctionnelles qui spécifient chacune un ensemble de services qu'elle doit fournir (principalement à la couche supérieure). Ce modèle définit 7 couches, allant de la couche *physique* à la couche *application* [62] :

1. **Couche physique :** Elle fournit les moyens matériels (électroniques, mécaniques, etc.) et fonctionnels permettant d'activer, maintenir et désactiver une connexion physique et une transmission de données sous format de bits. elle comprend tout moyen de transmission et de codage (câbles, ondes, modulation/démodulation, etc.).
2. **Couche liaison de données :** Elle fournit les moyens fonctionnels pour permettre la communication entre deux entités adjacentes du réseau partageant une connexion physique. Elle gère notamment l'adressage physique. Les PDU transmis sur cette couche sont généralement appelés *trames*.
3. **Couche réseaux :** Elle fournit des services de routage et permet la connexion entre entités ne se trouvant pas forcément dans le même réseau physique (et qui ne peuvent pas directement communiquer via les fonctions de la couche 2). Elle gère notamment l'adressage logique et permet aux couches supérieures de fonctionner indépendamment des problématiques de routage et de calcul de chemin. Les PDU transmis sur cette couche sont généralement appelés *paquets*.
4. **Couche transport :** Elle fournit aux couches supérieurs un service de transmission de données complètement transparent à la gestion des ressources réseau et des coûts associés à ces transmissions. Elle comprend des fonctions d'optimisation des ressources utilisées afin de répondre au mieux et à moindre coût aux besoins des entités de couches supérieures. Tous les protocoles définis dans la couche 4 sont de bout en bout.
5. **Couche session :** Elle gère la synchronisation des entités de la couche supérieure et permet de restaurer la communication à un état antérieur si des erreurs se produisent.
6. **Couche présentation :** Elle permet la conversion des données applicatives en données codées transmissibles entre les différentes entités. Elle permet également aux entités de la couche supérieure d'avoir une représentation commune et cohérente des données.
7. **Couche application :** Elle constitue le point d'accès aux services du réseau. Les fonctions de cette couche (ou applications) peuvent être tout service utile à l'utilisateur ainsi que les interfaces entre ce dernier et les services.

4.3.2 Exemple : le cas IP sur Ethernet

Une des instanciations les plus courantes de ce fonctionnement en couches est le cas IP sur *Ethernet*. Cette section présente brièvement les deux protocoles et la manière dont les paquets IP sont *encapsulés* dans des trames Ethernet.

4.3.2.1 Le protocole IP :

Le protocole IP se situe à la couche 3 (réseau) du modèle OSI. Il en existe plusieurs versions (principalement IPv4 et IPv6), nous nous intéressons ici à la version 4 (IPv4). Le protocole IPv4 gère l'adressage logique dans le réseau. Chaque entité communicante dispose d'une adresse IP unique qui l'identifie. Les données sont structurées en paquets et chaque paquet contient un en-tête et un champ de données. La taille maximale d'un paquet est de 64ko.

La figure 4.4 montre la structure d'un en-tête de paquet IPv4. Le lecteur intéressé par les fonctions des différents champs peut se référer aux standards définis par l'*Internet Engineering Task Force* (IETF) : [92, 16, 86, 110] entre autres.

Remarque. Le protocole IP est normalement défini dans le modèle TCP/IP [21]. Il se situe au niveau de la couche 2 de ce modèle. On peut néanmoins faire une correspondance grossière (avec quelques

21. TCP : Transmission Control Protocol.

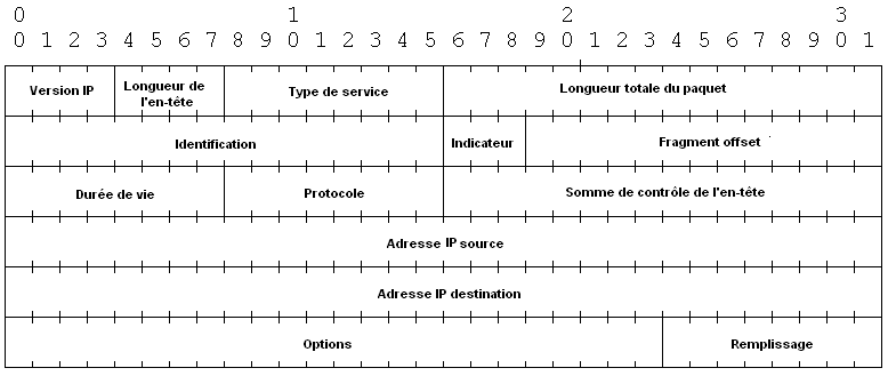


FIGURE 4.4 – En-tête d'un paquet IP [92]

inexactitudes) entre la couche 2 du modèle TCP/IP et la couche 3 du modèle OSI. Nous avons préféré présenter ces protocoles dans le modèle OSI car il est plus générique et ne correspond pas à une technologie donnée.

4.3.2.2 Le protocole Ethernet

Le protocole Ethernet se situe au niveau de la couche 2 du modèle OSI. Il en existe de nombreuses versions (Ethernet 802.3, Ethernet II, Ethernet 802.11, etc.). Nous nous intéressons au standard 802.3 [10]. Le protocole Ethernet gère l'adressage physique dans le réseau. Chaque entité dispose d'une adresse MAC [22] qui l'identifie. Les données sont structurées en trames. La taille minimum des données est de 46 octets. Si les données sont insuffisantes, ce champ est rempli de données inutiles pour atteindre la taille minimum. Usuellement, la taille maximum des données est de 1500 octets.

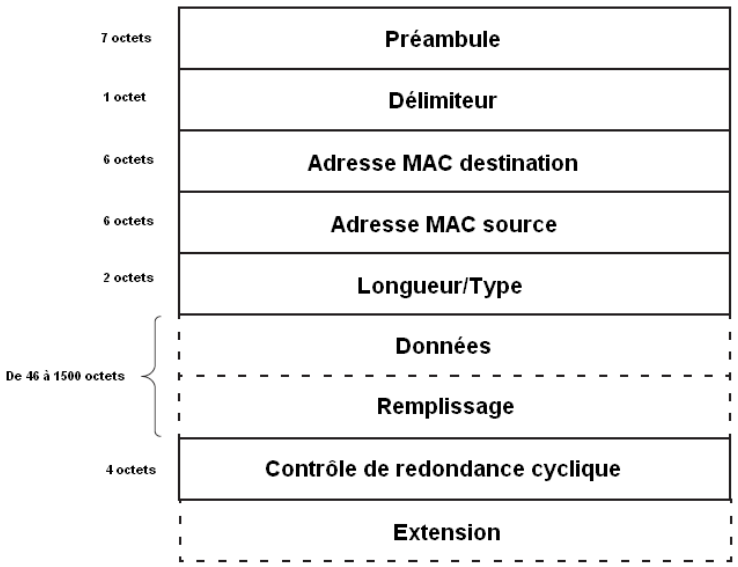


FIGURE 4.5 – Structure d'une trame Ethernet [10]

La figure 4.5 montre la structure d'une trame Ethernet. Le détail des fonctionnalités de chaque champ peut-être trouvé dans [10].

22. MAC : Media Acces Control. Adresse physique stockée dans les cartes réseau.

4.3.2.3 Encapsulation de paquets IP dans des trames Ethernet

Pour transporter des paquets IP sur un réseau Ethernet, les paquets IP sont encapsulés dans les trames. C'est-à-dire que chaque paquet IP (en-tête et données) est placé dans le champ "données" de la trame Ethernet. L'encapsulation de paquets IPv4 dans Ethernet est standardisée dans [59]. L'encapsulation de paquets IPv6 est standardisée dans [33].

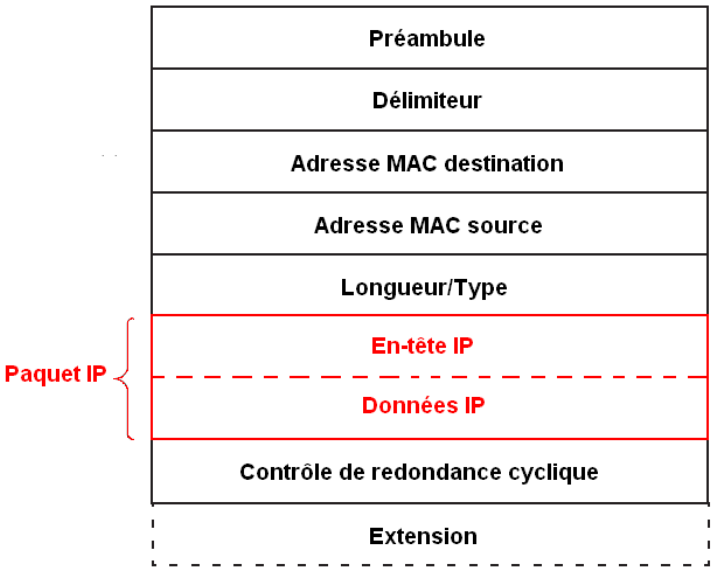


FIGURE 4.6 – Encapsulation d'un paquet IP dans une trame Ethernet.

La figure 4.6 montre un paquet IP encapsulé dans une trame Ethernet. Il est placé dans le champ "données" de la trame. Cependant, ce champ est limité à 1500 octets alors qu'un paquet IPv4 peut atteindre 64ko. Les deux solutions les plus courantes à ce problème de compatibilité sont les suivantes :

- Limiter la taille des paquets IP à 1500 octets. Cela peut se faire par paramétrage sur les couches supérieures (par exemple limiter la taille des segments TCP, ce qui aura pour effet de limiter la taille des paquets IP [93]).
- *Fragmenter* les paquets IP de taille supérieure à 1500 octets et encapsuler chaque fragment dans une trame.

4.4 Pseudo-Wire

Le protocole IP s'est largement généralisé ces vingt dernières années et occupait la plus large place dans les réseaux dès les années 1990. Néanmoins, cette convergence sur IP a posé la question de continuer à assurer certains services qui étaient transportés sur les couches basses (1 et 2), appelés *services natifs* par la suite. D'autant plus que si le protocole IP était le plus largement déployé, il n'était pas celui qui rapportait le plus haut revenu par bit [119] (en 2004). Les services sur Frame Relay. et les lignes privées sur TDM [23] rapportaient plus en moyenne. Qui plus est, énormément d'équipements déployés dans les réseaux publics avant la généralisation d'IP continuaient à fonctionner avec d'anciens protocoles, et il était utile de pouvoir les interconnecter via les nouveaux réseaux IP.

L'architecture Pseudo-Wire [26] a donc été proposée pour permettre l'émulation de protocoles des couches 1 et 2 (services natifs) sur des réseaux à commutation de paquets (*Packet-Switched Networks* - PSN), contrairement à ce que préconise le modèle OSI. Cette émulation permet d'assurer une connectivité de bout en bout entre plusieurs parties d'un réseau utilisant un protocole de couches basses, et ceci via un segment PSN (principalement IP ou MPLS [24]). Les PDU sont encapsulés dans des paquets pour traverser un segment IP ou IP-MPLS, puis désencapsulés à la sortie du segment. Cette émulation est transparente aux entités de couches basses se situant aux deux bouts de la connexion.

23. TDM : Time Division Multiplexing.
24. MPLS : MultiProtocol Label Switching

4.4.1 Architecture Pseudo-Wire

L'architecture Pseudo-Wire, définie dans [26], spécifie les fonctionnalités requises pour l'émulation de services natifs sur un réseau PSN. La figure 4.7 montre le modèle de référence d'un réseau Pseudo-Wire. Les deux *Customer Edges* (CE1 et CE2) font partie d'un réseau utilisant un service natif. Les *provider Edges* (PE1 et PE2) vont les connecter via un tunnel PSN dans lequel les PDU de CE1 et CE2 sont encapsulés dans les paquets du PSN. Cette émulation est transparente aux deux CE. En effet, les PDU sont transportés entre CE1 et CE2 comme s'ils étaient directement reliés par un service natif.

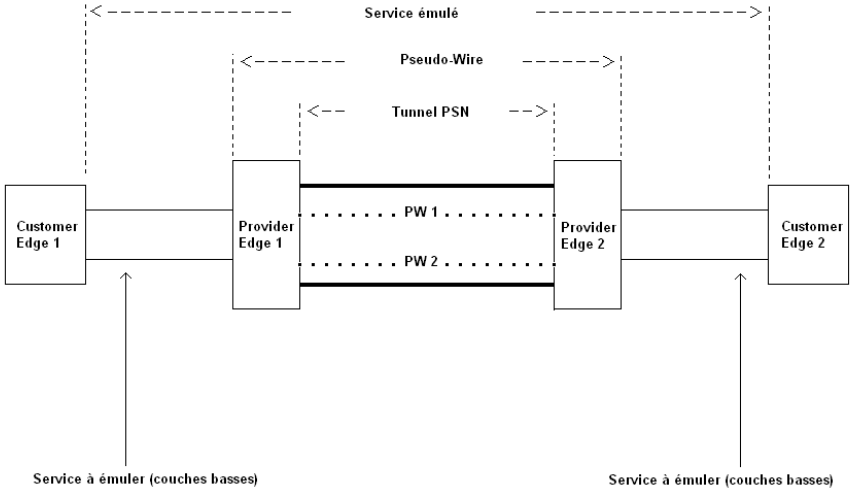


FIGURE 4.7 – Modèle de référence d'un réseau Pseudo-Wire [26].

Les PE sont en charge des fonctionnalités nécessaires à l'émulation, principalement :

- **Encapsulation :** Les PDU des protocoles émulés (trames Ethernet, cellules ATM [25], etc.) sont encapsulés dans les paquets PSN. Ils sont désencapsulés à l'autre extrémité du tunnel.
- **Transport :** Après l'encapsulation, les PDU du service natif sont acheminés via un tunnel PSN entre les deux PE.
- **Connexion :** Etablissement de la connexion entre les deux PE (échange d'identifiants, etc.).
- **Management :** Gestion de la signalisation entre les PE, gestion de la synchronisation, de l'ordre des paquets, etc.
- **Autres :** Prise en charge des fonctionnalités spécifiques au service natif (alarmes, messages *keep alive*, etc.).

L'encapsulation en elle-même consiste à placer le PDU du service natif dans le champ "donnée" des paquets du PSN. Néanmoins, plusieurs fonctionnalités qui ne sont pas habituellement assurées par le PSN doivent êtres préservées ou restaurées :

- **Séquencement :** Il consiste à attribuer un *numéro de séquence* aux PDU. Certains services natifs nécessitent que l'ordre d'arrivée des PDU soit le même que l'ordre d'émission, ce qui peut ne pas être garanti par un réseau PSN (notamment IP). Le séquencement permet donc de restituer l'ordre initial des PDU à l'arrivée, via une mémoire tampon. Il permet également de détecter la duplication de certains PDU (en constatant que deux PDU ont le même numéro de séquence). Enfin, le séquencement permet la detection de pertes de PDU si certains numéros de séquence manquent à l'arrivée. Le séquencement n'est à effectuer que si le service natif doit assurer une (ou plusieurs) de ces trois fonctionnalités.
- **Synchronisation :** Certains services natifs (par exemple TDM) sont synchrones. Cette caractéristique doit pouvoir être restituée au moment de la désencapsulation. Particulièrement, il doit y avoir un mécanisme permettant de synchroniser les horloges aux deux bouts du réseau. La même contrainte s'applique à l'envoi différé (laps de temps constant entre l'émission d'un PDU et sa réception).
- **Fragmentation :** Comme dans le cas d'IP sur Ethernet (section 4.3.2.3), il se peut que la taille

25. ATM : Asynchronus Transfer Mode

d'un PDU du service natif soit supérieure à la taille maximale d'un paquet PSN. Dans ce cas, la fragmentation du PDU doit se faire avant l'encapsulation dans plusieurs paquets. Le regroupement de ces paquets et la reconstitution du PDU initial se fait après la désencapsulation. Les techniques de fragmentation et de restauration sont détaillées dans [76].

4.4.2 Exemple d'encapsulation : Ethernet sur MPLS

La RFC 4448 [81] définit l'encapsulation d'une trame Ethernet dans MPLS au niveau d'un PE. La procédure d'encapsulation est la suivante :

1. Supprimer le préambule et le champ "contrôle de redondance cyclique" de la trame Ethernet.
2. Préfixer la trame résultante avec un label *control word*.
3. Préfixer la trame résultante avec un VC [26] label qui permet d'identifier le Pseudo-Wire (dans le cas où plusieurs Pseudo-Wires sont établis sur le même tunnel). Ce label est aussi appelé *Pseudo-Wire demultiplexer*.
4. Préfixer la trame résultante avec un ou plusieurs labels MPLS selon la manière dont le tunnel MPLS est configuré. Ces labels serviront au routage dans le segment MPLS.

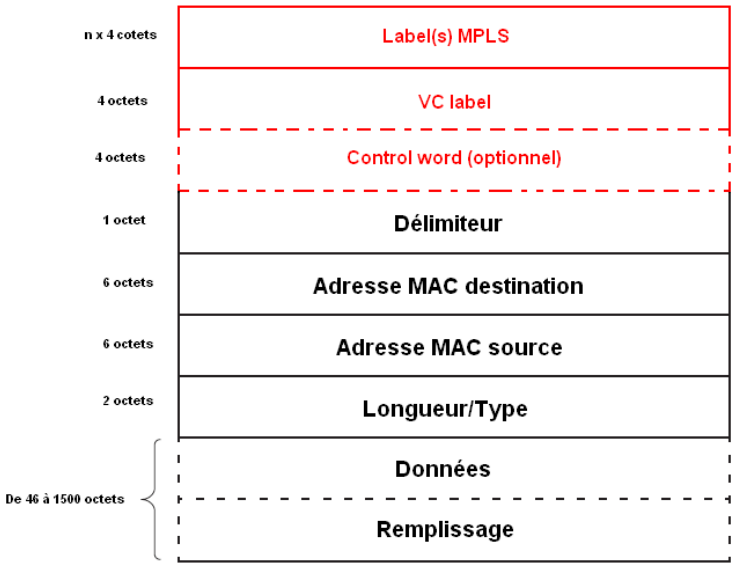


FIGURE 4.8 – Trame Ethernet encapsulée dans MPLS.

La figure 4.8 montre une trame Ethernet encapsulée dans MPLS. Le routage se fait grâce à la commutation des labels MPLS dans le tunnel. Chaque nœud intermédiaire dans le tunnel MPLS constatera le lien d'arrivée et le label en tête du paquet reçu, remplacera éventuellement le premier label par un autre et enverra le paquet résultant vers le prochain nœud du tunnel. Le PE d'arrivée enlèvera le label MPLS ainsi que le VC label (ou Pseudo-Wire demultiplexer label). Celui-ci permet d'identifier à quelle connexion Pseudo-Wire appartient la trame dans le cas où plusieurs connexions Pseudo-Wire sont établies sur le même tunnel.

4.4.2.1 Le *control word*

Le *control word* est un label spécifique qui permet d'avoir des informations sur le Pseudo-Wire. il est optionnel dans le cas d'Ethernet sur MPLS (mais peut être obligatoire pour d'autres encapsulations). Il permet, entre autres, de traiter les trames individuellement, de les réordonner grâce au champ "numéro de séquence". La figure 4.9 montre la structure de ce label. Les 4 premiers bits doivent être à 0 pour éviter la confusion avec le début d'un paquet IPv4 ou IPv6. Le champ "flags" ainsi que les deux bits

26. VC : Virtual Circuit.

suivants sont réservés dans le cas d'Ethernet sur MPLS. Ils doivent être mis à 0. Le champ "taille" doit être mis à 0 si la taille de la trame est supérieure à 64 octets. Enfin, le dernier champ peut être utilisé comme numéro de séquence pour pouvoir réordonner les trames à l'arrivée.

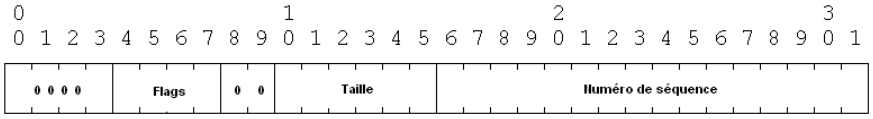


FIGURE 4.9 – Structure du label *control word* [80].

Plusieurs autres encapsulations ont été définies. On peut citer notamment TDM sur IP [99], Frame Relay sur MPLS [78], ATM sur MPLS [77], etc.

4.4.3 Le Pseudo-Wire multisegment

L'établissement d'un tunnel Pseudo-Wire entre chaque couple de CE aboutit à un graphe complet de tunnels. Le nombre de tunnels et les configurations manuelles nécessaires pour leur établissement devient vite prohibitif. Pour pallier ce problème, l'architecture Pseudo-Wire multisegment [23] propose d'étendre l'architecture Pseudo-Wire initiale à plusieurs segments PSN.

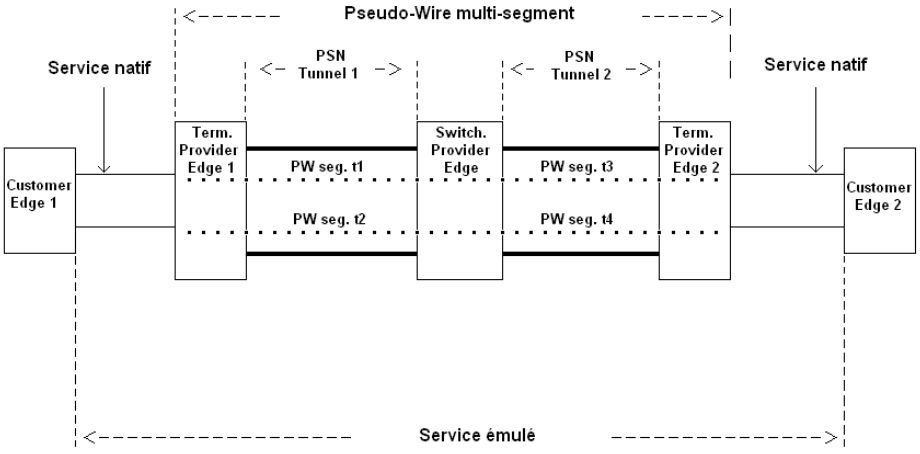


FIGURE 4.10 – Modèle de référence d'un réseau Pseudo-Wire multisegment [23].

La figure 4.10 montre le modèle de référence d'un réseau Pseudo-Wire multisegment. Les *Terminating Provider Edges* (TPE) ne sont pas directement reliés par un tunnel PSN, mais par plusieurs tunnels établis entre eux et un (ou plusieurs) *Switching Provider Edge* (SPE). Les SPE permettent de connecter les différents segments qui, mis bout à bout, relient les deux TPE. Cette architecture facilite notamment l'établissement de Pseudo-Wire traversant plusieurs domaines. En effet, les différents segments (ou tunnels) peuvent appartenir à différents domaines, comme le montre la figure 4.11. Ici, les CE sont reliés par trois segments PSN. Le premier segment (Seg. 1) appartient au domaine 1, le troisième segment (Seg. 3) appartient au domaine 2, et le deuxième segment (Seg. 2) est l'interconnexion entre les deux domaines.

Il est à noter que les différents segments d'un Pseudo-Wire multisegment ne sont pas forcément de même type. Certains tunnels peuvent être IP, d'autres MPLS, etc. Entre deux segments, c'est aux SPE de convertir (ou *mapper*) les paquets entrants avant de pouvoir les envoyer sur le segment suivant. Il se peut également que l'interconnexion entre deux domaines fasse intervenir des encapsulations de protocoles, les encapsulations du Pseudo-Wire se trouveront donc imbriquées dans d'autres encapsulations. D'autres possibilités existent au niveau de l'interconnexion entre deux domaines. Un SPE se trouvant en bordure de domaine peut desencapsuler les PDU du service natif et les transmettre à un SPE de bordure du domaine suivant qui les encapsulera de nouveau avant de les envoyer dans le segment suivant. Ainsi,

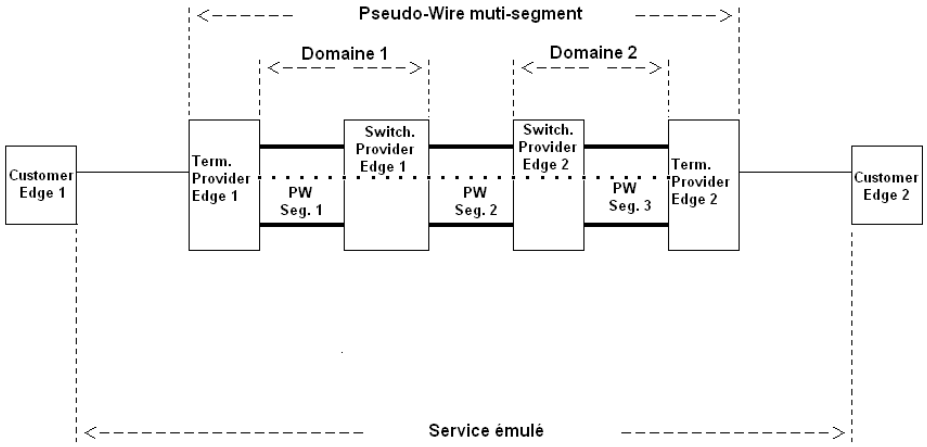


FIGURE 4.11 – Modèle de référence d'un réseau Pseudo-Wire multisegment sur plusieurs domaines [23].

l'interconnexion entre les deux domaines ne se fera pas via un PSN mais via le service natif. Les différents types d'interconnexion de segments Pseudo-Wire sont détaillés dans [79].

4.4.4 Calcul de chemin dans un réseau Pseudo-Wire multisegment

Calculer un chemin dans un réseau Pseudo-Wire multisegment revient à sélectionner les SPE et les tunnels qui doivent en faire partie. Cette sélection, ainsi que la configuration des nœuds se font manuellement. Disposer d'un algorithme de calcul de chemins comme spécifié dans la section 4.2.3 permettrait en partie d'automatiser ces tâches. Cho *et al.* [31] soulignent l'importance d'un tel algorithme et précisent qu'il n'y a pas de solution évidente à l'heure actuelle.

Le réseau pseudo-Wire peut être modélisé comme un réseau dans lequel coexistent plusieurs protocoles et où certains noeuds ont des capacités d'encapsulation et de désencapsulation. Cela revient au modèle abstrait présenté sur la figure 4.3.

La fonction objectif à minimiser serait l'une des trois présentées dans la section 4.2.3. Celle qui correspond au nombre d'encapsulations est très importante en raison de la fragmentation, du réordonnancement et du regroupement des PDU. La résolution de ce problème sera l'objet du chapitre suivant.

4.5 Problèmes similaires

Le problème de calcul de chemins décrit en section 4.4.4 ne se retrouve pas uniquement dans le cas du Pseudo-Wire multisegment. En effet, on retrouve les problématiques d'encapsulation dans divers contextes et avec plusieurs autres technologies réseau. Cette section expose quelques problèmes de calcul de chemins similaires.

4.5.1 Calcul de chemins dans les réseaux multicouches

Dans les réseaux multirégions et multicouches (*Multi-Region and Multi-Layer Networks*,- MRN/MLN) [107], les différentes régions ont des capacités de commutation sur différentes couches. Par exemple, certaines régions font de la commutation avec Ethernet (couche 2), d'autres font de la commutation de longueurs d'onde sur les réseaux optiques (couche 1), d'autres encore peuvent faire les deux. Il peut y avoir des fonctions d'adaptation à l'intérieur des régions comme en bordure de région.

Le calcul de chemin dans un tel réseau doit prendre en compte les fonctions d'adaptation, ainsi que la compatibilité entre les protocoles des différentes couches. Contrairement à l'architecture Pseudo-Wire, le modèle OSI est respecté car des couches basses ne sont pas émulées sur des couches supérieures. Néanmoins, n'importe quel protocole de couche n ne peut pas être transporté sur n'importe quel protocole de couche $n-1$. Les fonctions d'adaptation peuvent êtres vues comme des encapsulations possibles d'un protocole de couche n dans un protocole de couche $n-1$.

4.5.2 Calcul de chemins dans les NREN

Les *National Research and Education Networks* (NREN) sont des réseaux principalement dédiés à l'enseignement et à la Recherche. Ils ont la particularité d'être *hybrides*, c'est-à-dire qu'ils permettent en même temps une connectivité IP et une connectivité sur des couches plus basses (connectivité Ethernet, par exemple) [40].

Dans un NREN, certains domaines universitaires sont reliés à leur fournisseur d'accès via Ethernet. Mais pour les connecter, il faut passer par des points d'interconnexion entre domaines qui utilisent le plus souvent une technologie optique (SONET/SDH [27] par exemple). Il faut donc pouvoir encapsuler Ethernet dans SONET/SDH. La désencapsulation doit se faire avant d'arriver au site destinataire. Or, il existe plusieurs standards différents pour ces encapsulations, il faut donc veiller à ce que chaque désencapsulation soit appliquée en correspondance avec la bonne encapsulation.

4.5.3 Tunnels dans Internet

Comme expliqué précédemment, un tunnel est une portion d'un chemin où le protocole d'origine est encapsulé dans un autre à l'entrée et désencapsulé à la sortie. Les tunnels jouent un rôle de plus en plus important dans Internet. Outre l'architecture Pseudo-Wire, on peut citer comme exemples :

- **IPsec :** Encapsule des paquets IP dans d'autres paquets IP afin de masquer l'adresse IP réelle du destinataire [65].
- **IPv6 sur IPv4 :** Relie des îlots IPv6 via des tunnels IPv4 [29].
- **LISP** [28] **:** Permet de séparer l'identification et la localisation des utilisateurs. Les paquets IP sont encapsulés par un routeur, acheminés jusqu'au domaine destinataire puis désencapsulés par un autre routeur avant d'être envoyés au destinataire final [42].

Touch et Townsley [111] identifient les principaux points à traiter concernant l'établissement de tunnels : la fragmentation (il peut y avoir incompatibilité entre la taille des paquets qu'on encapsule et la taille des paquets du tunnel) et la signalisation (la transmission des messages sur l'état du réseau et leur interférence avec les tunnels).

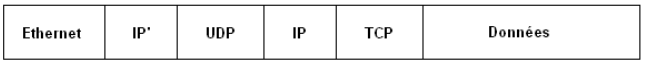


FIGURE 4.12 – Paquet IP encapsulé dans un segment UDP, lui-même encapsulé dans un paquet IP, lui-même encapsulé dans une trame Ethernet [111].

La figure 4.12 montre plusieurs encapsulations imbriquées (IP sur UDP [29] sur IP sur Ethernet). Si toutes les encapsulations se font au nœud émetteur et toutes les désencapsulations se font au niveau du nœud destinataire, le calcul de chemins ne pose pas de problème. Cependant, si les encapsulations ou les désencapsulations sont distribuées dans le réseau, le calcul de chemins doit prendre en compte les changements de protocoles et la compatibilité entre eux.

4.6 Conclusion

Devant l'hétérogénéité des protocoles utilisés par les différent domaines, l'encapsulation de protocoles est utile pour l'établissement de chemins inter-domaine de bout en bout. L'encapsulation consiste à placer les unités d'informations d'un protocole dans les unités d'informations d'un autre. Le calcul de chemins prenant en compte ces encapsulations est complexe et ne peut être effectué par des algorithmes de calcul de chemins classiques. La structuration des réseaux en couches par le modèle OSI définit un cadre naturel à l'encapsulation de protocoles. Néanmoins, l'architecture Pseudo-Wire, qui est largement répandue aujourd'hui, définit des encapsulations qui ne respectent pas le modèle OSI. Ces encapsulations permettent principalement d'émuler des protocoles de couches basses (Ethernet, Frame Relay, ATM, etc.) sur des réseaux à commutation de paquets (principalement IP et MPLS). Plusieurs segments de Pseudo-Wire

27. SONET/SDH : Synchronous Optical NETwork/Synchronous Digital Hierarchy.
28. LISP : Locator I/D Separator Protocol.
29. UDP : User Datagram Protocol.

peuvent être mis bout à bout dans le réseau pour établir un chemin. Le calcul d'un tel chemin nécessite de prendre en compte les contraintes de compatibilité ainsi que l'emplacement des fonctions d'encapsulation dans le réseau. Cette problématique ne se retrouve pas seulement dans les réseaux Pseudo-Wire. D'autres types de réseaux (MRN/MLN, NREN, etc.) imposent les mêmes contraintes pour calcul de chemins, de même que l'établissement de tunnels (sur n'importe quelles couches) dans Internet.

Dans ce chapitre, nous nous sommes proposé de présenter les notions d'encapsulation et de désencapsulation de protocoles de manière abstraite. Nous avons présenté la structuration en couches des réseaux ainsi qu'un exemple classique d'encapsulation. Nous avons présenté l'architecture Pseudo-Wire et décrit plusieurs encapsulations qu'elle définit, ainsi que l'architecture Pseudo-Wire multisegment et la problématique de calcul de chemins dans cette architecture. Enfin, nous avons donné d'autres exemples où le calcul de chemins doit prendre en compte les fonctions d'encapsulation et de désencapsulation. Dans le chapitre suivant, nous décrirons ce problème de manière formelle et proposerons des solutions d'abord sans contrainte de bande passante, puis avec des contraintes de QoS.

5

Calcul de chemins dans les réseaux multicouches

5.1 Introduction

Dans le chapitre précédent, nous avons évoqué les aspects hétérogènes et multicouches des réseaux actuels. Nous avons vu que plusieurs architectures proposaient des fonctions d'encapsulation et de désencapsulation de protocoles pour permettre la communication de bout en bout malgré cette hétérogénéité.

Le principal problème qui se pose est le calcul de chemins dans un tel réseau. En effet, il ne suffit pas que deux nœuds soient physiquement connectés via un chemin (ensemble de lien adjacents) pour que ces nœuds puissent communiquer. En plus de la connectivité physique, il faut assurer une *continuité protocolaire* entre les deux nœuds, c'est-à-dire que chaque couple de nœuds aux deux extrémités d'un lien doit utiliser le même protocole (même si le protocole en entrée dans un nœud n'est pas forcément le même qu'en sortie). Il faut donc prendre en compte les capacités d'encapsulation et de désencapsulation des nœuds et utiliser ces capacités à l'endroit adéquat pour assurer la continuité protocolaire. Les contraintes générées par la continuité protocolaire sont appelées *contraintes de compatibilités*. Un exemple d'une telle contrainte est que si un nœud reçoit un protocole x et envoie un protocole y, il faut que ce nœud ait la capacité soit d'encapsuler x dans y, soit d'extraire y à partir de x. Dans le second cas, il faudrait qu'un autre nœud ait encapsulé y dans x en amont dans le chemin. Un chemin qui respecte les contraintes de compatibilité sera appelé chemin *faisable*.

Dans ce contexte, le plus court chemin entre deux nœuds peut avoir des caractéristiques non triviales. En effet, les chemins faisables n'ont pas une sous-structure optimale. Les sous-chemins du chemin faisable le plus court entre deux nœuds ne sont pas forcément des chemins les plus courts, car ils ne sont pas forcément faisables. En prenant trois nœuds, U, V et W, il est possible de construire un réseau où il n'y a pas de chemins faisables entre U et V, ni entre V et W, mais où existe un chemin faisable entre U et W passant par V. Une autre caractéristique non triviale des chemins faisables les plus courts est qu'ils peuvent comporter des boucles. De ce fait, la contrainte de bande passante n'est plus élagable [30], car on ne sait pas à l'avance combien de fois un chemin passera par un lien en particulier.

Dans le cadre de cette thèse, notre but est de calculer des chemins prenant en compte les encapsulations et désencapsulations et assurant une certaine QoS. Le problème est NP-complet car il comprend le sous-problème de calcul de chemin avec QoS et sans encapsulations, qui est déjà NP-complet [113]. Le problème reste NP-complet même si l'on ne garde que la contrainte de bande passante et les contraintes de compatibilité [67]. Cependant, même en relaxant la contrainte de bande passante, aucune solution triviale n'apparaît. Les algorithmes de calcul de chemins classiques ne peuvent résoudre le problème car ils ne prennent pas en compte les contraintes de compatibilité et se basent sur la sous-structure optimale des chemins les plus courts, sous-structure que nous perdons à cause des contraintes de compatibilité.

Pour résoudre ce problème et concevoir la première solution polynomiale au calcul de chemins faisables (sans contrainte de bande passante, ni QoS), nous caractérisons les chemins faisables à l'aide d'outils de la Théorie des Langages. Plus spécifiquement, nous montrons que les séquences de protocoles associées aux chemins faisables forment un langage à contexte libre. Nous modélisons donc le réseau multicouche en automate à pile, automate que nous convertissons ensuite en grammaire à contexte libre. Cette grammaire

30. *Prunable* en anglais.

génèrera la séquence de protocoles du chemin faisable le plus court, séquence qui nous permettra de calculer ce chemin. De plus, nous optimisons la longueur du chemin selon deux métriques : le nombre de liens ou le nombre de fonctions d'encapsulation dans le chemin. Généralisant cette approche, nous proposons une solution au calcul de chemin faisable le plus court sous n'importe quelle mesure additive. Cette approche ne s'applique malheureusement pas quand nous ajoutons la contrainte de bande passante. Nous améliorons le résultat de NP-complétude de [67] en montrant que le problème reste NP-complet même avec deux protocoles uniquement. Nous présentons brièvement l'algorithme SAMCRA [31], proposé par Kuipers et Van Mieghem [82], qui calcule des chemins sous contraintes de QoS. Cet algorithme a une complexité moyenne polynomiale (sa complexité dans le pire des cas reste exponentielle, étant donné que le problème est NP-complet). Nous adaptons cet algorithme et le généralisons pour qu'il prenne en compte les contraintes de compatibilité et permette ainsi de calculer des chemins faisables avec garantie de QoS.

Ce chapitre est organisé comme suite : la section 5.2 présente le problème ainsi que l'état actuel de la recherche sur le sujet et détaille notre approche ; la section 5.3 présente le modèle de réseau multicouche que nous utilisons ainsi que quelques définitions utiles ; la section 5.4, qui est le cœur de ce chapitre, détaille notre méthode pour calculer le chemin faisable le plus court en nombre de lien ou de fonctions d'encapsulation, les algorithmes utilisés sont prouvés et leur complexité étudiée ; la section 5.5 généralise les résultats de la section précédente au calcul du chemin le plus court sous n'importe quelle mesure additive sur les liens et les fonctions d'adaptation ; la section 5.6 présente le même problème mais avec une contrainte de bande passante et montre qu'il est NP-complet avec deux protocoles ; la section 5.7 présente le problème général sous contraintes de QoS, l'algorithme SAMCRA ainsi que notre adaptation de celui-ci pour la prise en compte des contraintes de compatibilité ; enfin, la section 5.8 conclut le chapitre.

Ce chapitre est basée sur les articles [W2], [C2] et [J1] publiés dans le cadre de cette thèse.

5.2 Calcul de chemins dans les réseaux multicouches

5.2.1 Caractéristiques du chemin le plus court dans un réseau multicouche

Dans cette section, nous allons exposer de manière non formelle et à travers des exemples certaines des caractéristiques des chemins dans les réseaux multicouches.

5.2.1.1 Chemins avec boucles

Il apparait que le chemin faisable le plus court (voire le seul chemin faisable) entre deux nœuds dans un réseau multicouche peut comporter des boucles.

La figure 5.1 représente un réseau multicouche. Les capacités d'encapsulation et de désencapsulation de chaque nœud sont notées à côté de celui-ci. Par exemple, le nœud U_1 a la capacité d'encapsuler les paquets du protocole a dans des paquets de protocole b. Le nœud U_2 a la capacité de transmettre passivement (sans modification) les paquets du protocole b. Nous cherchons le chemin faisable le plus court du nœud S vers le nœud D, sachant que S transmet des paquets de protocole a. Le chemin direct (en rouge) consistant à passer par les nœuds U_1,U_2,U_4,U_5,U_6 n'est pas faisable car, arrivés au nœud U_5, des paquets de b doivent être extraits des paquets de a. Or, comme on peut le voir sur la pile de protocole au niveau de l'entrée de U_5, les paquets de a ne contiennent pas des paquets de b.

La figure 5.2 représente le seul chemin faisable sur le même réseau. Ce chemin comporte une boucle, il passe deux fois par le lien (U_1,U_2). L'état de la pile de protocole est indiqué sous chaque lien. le protocole en sommet de pile est le protocole courant, les autres protocoles sont les protocoles encapsulés. Cette boucle est nécessaire pour emmagasiner des encapsulations qui seront nécessaires au passage par les nœuds U_5 et U_6.

Cette propriété rend la satisfaction de la contrainte de bande passante non triviale, car **on ne sait pas à l'avance combien de fois un même lien sera parcouru par un chemin.** Nous discuterons de l'impact de cette propriété sur la classe de complexité du problème de calcul de chemins en section 5.6.2.

31. SAMCRA : Self-Adaptive Multiple Constraints Routing Algorithm.

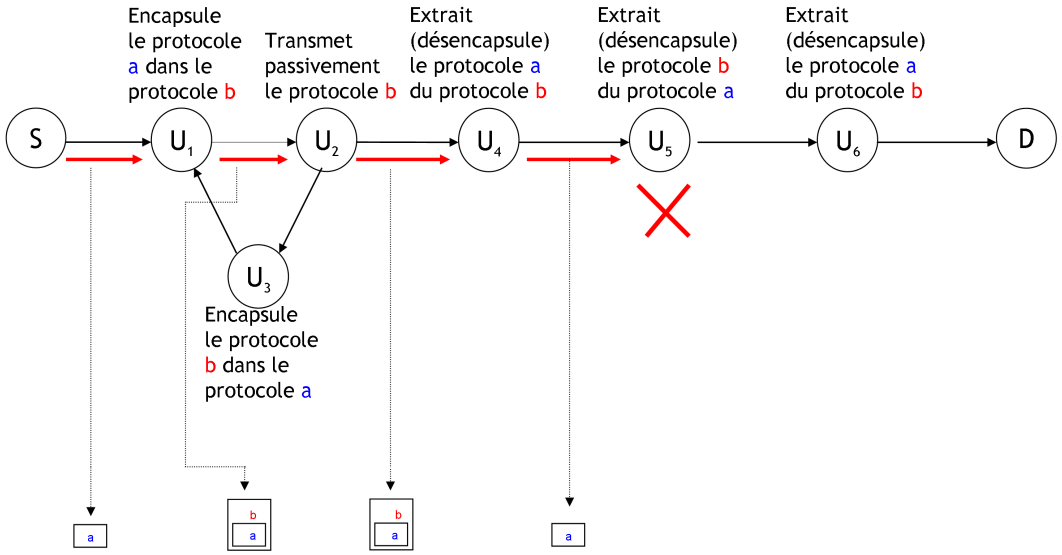


FIGURE 5.1 – Chemin direct mais non faisable dans un réseau multicouche. Les flèches indiquent l'état de la pile de protocoles à certains points du réseau.

5.2.1.2 Absence de sous-structure optimale

On dit d'un problème global qu'il possède une sous-structure optimale si la solution optimale à ce problème peut se décomposer en sous-solutions optimales aux sous-problèmes composant le problème global. Par exemple, le plus court chemin dans un graphe (orienté ou non) a une sous-structure optimale. En effet, tous les sous-chemins du chemin le plus court sont eux-mêmes des chemins les plus courts entre leurs deux extrémités. C'est cette propriété qui permet l'utilisation de la programmation dynamique dans les algorithmes de calcul du plus court chemin (Dijkstra [39], Floyd-Warshall [44, 114], etc.).

Les chemins faisables ne possèdent pas cette propriété, un sous-chemin d'un chemin faisable n'est pas forcément faisable. Pour s'en convaincre, il suffit d'observer le chemin faisable sur la figure 5.2. Le chemin entier entre S et D est faisable. Cependant, si on considère le sous-chemin entre S et U_5, on voit bien qu'il n'est pas faisable car la pile de protocole contient encore plusieurs protocoles encapsulés au niveau du nœud destination. Ils n'ont pas été désencapsulés.

5.2.2 Algorithmes de calcul de chemins dans les réseaux multicouches

5.2.2.1 Calcul de chemins dans les réseaux optiques hétérogènes

Le problème du calcul de chemins dans les réseaux hétérogènes s'est d'abord posé au niveau optique. En effet, différentes longueurs d'onde coexistent dans les réseaux optiques, certains nœuds ayant la capacité de convertir certaines longueurs d'onde en d'autres. Une des contraintes posées étant la *continuité de longueur d'onde* : le long d'un chemin, tout nœud traversé doit l'être avec une longueur d'onde qu'il peut traiter, et un changement de longueur n'intervient à un nœud que si ce nœud a la capacité de conversion nécessaire. Cette contrainte est proche de la continuité protocolaire (et des contraintes de compatibilité qu'elle induit). Dans les deux cas, il s'agit de convertir une longueur d'onde (qui peut-être vue comme un protocole de couche 1) en une autre au bon endroit, de façon à ce que tous les nœuds du chemin puissent traiter les longueurs d'onde qu'ils reçoivent. La principale différence est qu'il n'y a pas d'encapsulation, une fois le changement de longueur d'onde opéré, il n'est pas nécessaire d'effectuer la conversion inverse plus loin dans le chemin.

Chlamtac *et al.* [30] ont proposé un nouveau modèle et différents algorithmes permettant de calculer des chemins sous contrainte de continuité de longueur d'onde dans un réseau optique. Leur modèle, appelé *Wavelength Graph Model*, permet d'exprimer les différentes contraintes liées aux longueurs d'onde

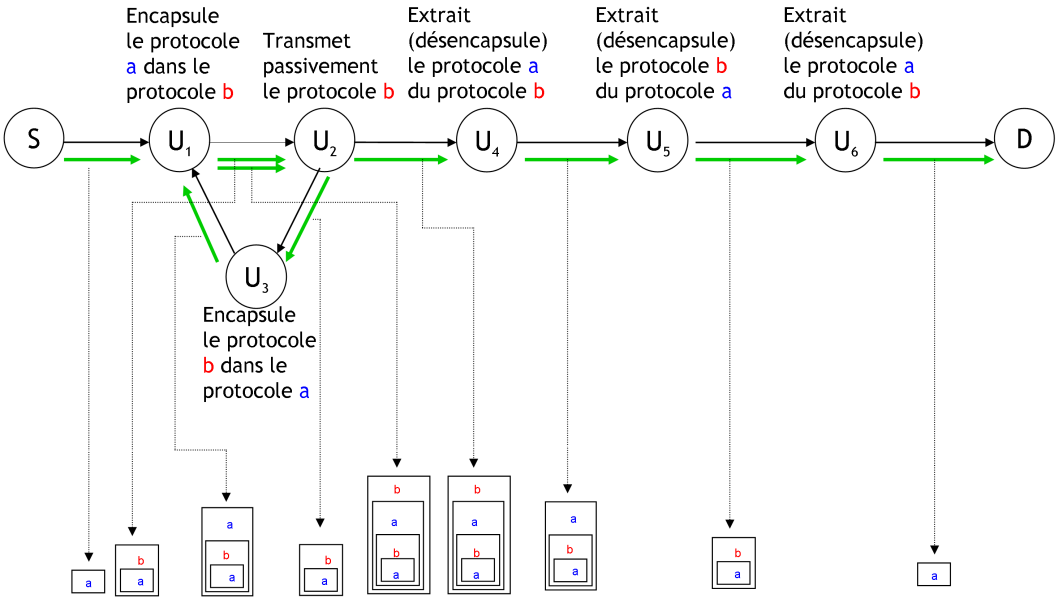


FIGURE 5.2 – Chemin faisable contenant une boucle dans un réseau multicouche.

qu'un graphe classique modélisant la topologie du réseau ne peut pas exprimer. Zhu *et al.* [122] ont traité le même problème dans les réseaux optiques mais en y ajoutant des contraintes liées au *traffic grooming*[32], ainsi que des contraintes de compatibilité sur deux couches (la couche optique et la couche accès). Là encore, un nouveau modèle, appelé *Auxiliary Graph Model* est proposé. Ce modèle permet également d'exprimer des contraintes de continuité de longueurs d'onde et de compatibilité qu'un graphe représentant la topologie du réseau ne peut pas exprimer. Yao et Ramamurthy [120] ont simplifié ce modèle et proposé d'autres approches pour le traffic grooming.

Le point commun de ces travaux est qu'ils proposent de nouveaux modèles où un équipement réseau (nœud physique) n'est plus modélisé par un seul nœud dans un graphe. De fait, le graphe ne représente plus simplement la topologie du réseau, mais également les différentes technologies et protocoles employés. Un équipement réseau correspond à différents nœuds dans le modèle, chaque nœud étant indexé par une technologie ou un protocole. L'existence d'un lien entre deux nœuds ne dépend pas uniquement de l'existence d'un lien physique entre eux, mais dépend également de la compatibilité des technologies ou protocoles utilisés par ces nœuds. Ainsi, un simple algorithme de calcul de chemin dans ce modèle permet de connaître la liste des nœuds traversés par ce chemin, mais également la liste des protocoles et technologies utilisés.

5.2.2.2 Calcul de chemins dans les réseaux multicouches

Le fait de gérer des contraintes de continuité sur plusieurs couches complique la tâche de calcul de chemins. En effet, il faut assurer la continuité de longueur d'onde (couche optique), la continuité de label MPLS (couche « 2,5 »)[33], etc. De plus, n'importe quelle technologie de couche 2 (Ethernet, par exemple), ne peut pas être transportée sur n'importe quelle technologie de la couche 1. Gong et Jabbari [51] ont proposé un modèle appelé *Channel Graph Model* qui permet de modéliser ces différentes contraintes et calculer le plus court chemin. Là encore, un équipement physique est modélisé par plusieurs nœuds, chacun indexé d'une technologie ou d'un protocole. Les liens horizontaux dans le graphe représentent des changements de technologies sur une même couche (conversion de longueur d'onde sur la couche optique,

32. Le *traffic grooming* est l'opération de regrouper plusieurs petits flux de communication au sein d'unités plus grandes, ces unités étant traitées plus tard comme des blocs indivisible. Le but de cette opération est de minimiser le traitement des données en augmentant leur granularité.

33. MPLS se trouvant entre la couche 2 et la couche 3 du modèle OSI, on le qualifie parfois de protocole de couche « 2,5 ».

par exemple). Les liens verticaux représentent des changements de couche (ajout d'un label MPLS sur une trame Ethernet, par exemple). Si un coût est attribué aux liens, alors l'application d'un algorithme de calcul du plus court chemin sur ce graphe permet de calculer un chemin de coût minimum. Cette approche ne permet cependant pas de résoudre notre problème. D'abord, elle ne permet de modéliser que des encapsulations classiques (couche 2 dans couche 1 par exemple), et ne permet pas de modéliser les encapsulations de type Pseudo-Wire (où l'on ne prend pas en compte la hiérarchie des couches ni le modèle OSI, et où les protocoles de couches basses peuvent être encapsulés dans des protocoles de couches plus hautes) car le modèle est statique et chaque niveau correspond à une couche bien définie. De plus, cette méthode nécessite un nombre de couches fixé à l'avance, ce qui implique un nombre d'encapsulations imbriquées fixé. Elle ne permet pas de modéliser une situation où un protocole x est encapsulé dans un protocole y, lui-même de nouveau encapsulé dans le protocole x car il faudrait représenter la même couche deux fois dans le modèle, ce qui n'est pas permis. De plus, la structure statique du modèle impose un ordre dans les encapsulations (les protocoles de couches hautes sont encapsulés dans des protocoles de couches plus basses. Faire une encapsulation dans l'autre sens est impossible).

Dans sa thèse, Freek Dikstra [34] [40] a étudié le problème de calcul de chemins dans un réseau multicouche. Cette thèse traite différentes architectures, mais pas l'architecture Pseudo-Wire. L'auteur définit les fonctions d'encapsulation et de désencapsulation dans le contexte de la recommendation ITU-T G.805, où les encapsulations se font uniquement d'une couche $n+1$ dans une couche n. Néanmoins, les solutions qu'il propose peuvent être généralisées aux encapsulations telles que nous les définissons (indépendamment de la hiérarchie des couches). Il affirme que les modèles classiques de graphes (représentant les topologies) ne peuvent correctement exprimer le problème. Il propose donc plusieurs modèles de graphes représentant les protocoles et technologies sur chaque couche. Certains de ces modèles ont une taille exponentielle en fonction de la topologie du réseau (ce qui explique que les algorithmes utilisés ensuite soient polynomiaux en fonction de la taille des nouveaux graphes). Cette thèse contient de nombreux exemples où le chemin le plus court dans un réseau multicouche comporte des boucles. Contrairement à notre travail, le problème traité par Dijkstra l'est uniquement sous contrainte de bande passante. Ainsi, dans un travail conjoint avec Kuipers [67], les auteurs montrent que le problème est NP-complet et proposent un algorithme de recherche en largeur qui explore tous les chemins jusqu'à en trouver un faisable et respectant une contrainte de bande passante donnée. Notre solution est beaucoup plus générale car elle traite le calcul de chemins sans contrainte de bande passante, avec contrainte de bande passante et avec contraintes de QoS. De plus, nous traitons plusieurs fonctions objectifs : nombre de liens, nombre d'encapsulations et mesure additive sur les liens et les fonctions d'adaptation.

5.2.2.3 Calcul de chemins sous contraintes de langages

Les travaux de Barrett *et al.* [21] sont l'approche théorique qui se rapproche le plus de la nôtre. Bien que la motivation pratique de ce travail et son domaine d'application soient les systèmes et modes de transport, la formalisation du problème se rapproche fortement du calcul de chemins dans les réseaux multicouches. En effet, leur modèle est un graphe dont les arêtes sont étiquetées par les symboles d'un alphabet donné Σ. On définit $l(C)$ comme étant la concaténation des étiquettes des arêtes le long du chemin C. Le problème est de trouver le plus court chemin C^* entre deux nœuds donnés dans le graphe tel que $l(C^*)$ appartient à un certain langage formel $\mathcal{L}$.

Leurs résultats montrent que si le chemin doit être simple (sans répétition de nœuds ni d'arêtes), le problème est NP-difficile quelque soit la nature du langage $\mathcal{L}$, sauf si c'est un langage fixe fini (auquel cas le problème est polynomial). Ce cas se rapproche du problème de calcul de chemins sous contrainte de bande passante et est à mettre en parallèle avec la NP-complétude de ce dernier. En effet, dans les réductions montrant la NP-complétude de ce dernier problème, la contrainte de bande passante impose généralement un chemin simple pour ne pas passer plusieurs fois par un même lien (et ainsi épuiser sa bande passante).

Cependant, si le chemin n'est pas forcément simple, le problème est polynomial pour plusieurs classes de langages (donnés par leur grammaire) : langages réguliers, langages à contexte libre, etc. Vu notre caractérisation des chemins faisables par un langage à contexte libre, le cas où $\mathcal{L}$ est donné par une grammaire à contexte libre se rapproche de notre problème (sans contrainte de bande passante). L'algorithme de calcul est un programme dynamique qui utilise les propriétés récursives des grammaires à contexte libre sous forme normale de Chomsky.

34. A ne pas confondre avec Edsger W. Dijkstra.

Bien que proche de la nôtre, cette approche n'est cependant pas totalement similaire et ne peut être appliquée à notre problème. En effet, dans le problème de Barrett *et al.* [21], la grammaire à contexte libre donnée est totalement indépendante du graphe. Ce qui reviendrait à dire qu'il suffirait qu'un chemin C ait la bonne suite d'étiquettes $l(C)$ pour qu'il soit faisable. Cette condition n'est pas suffisante car les fonctions d'encapsulation et de désencapsulation sont placées sur des nœuds bien spécifiques et ne peuvent être utilisées n'importe où dans le réseau. Dans notre approche, le réseau lui-même est converti en automate à pile, puis en grammaire à contexte libre. Cette grammaire dépend du langage qui caractérise les chemins faisables, mais également de la topologie du réseau.

5.2.3 Notre approche

Constatant qu'il n'y a avait pas de solution évidente pour le calcul de chemins faisables sans autre contrainte. Nous nous intéressons d'abord à ce problème et relaxons toute contrainte additionnelle. La caractérisation de la séquence de fonctions d'encapsulation et de désencapsulation d'un chemin faisable comme étant un langage bien parenthésé nous inspire une solution basée sur des outils de la Théorie des Langages. En effet, il s'avère que la séquence de protocoles d'un chemin faisable est également un langage à contexte libre. Retrouver la séquence de protocole la plus courte permet donc de retrouver le chemin le plus court.

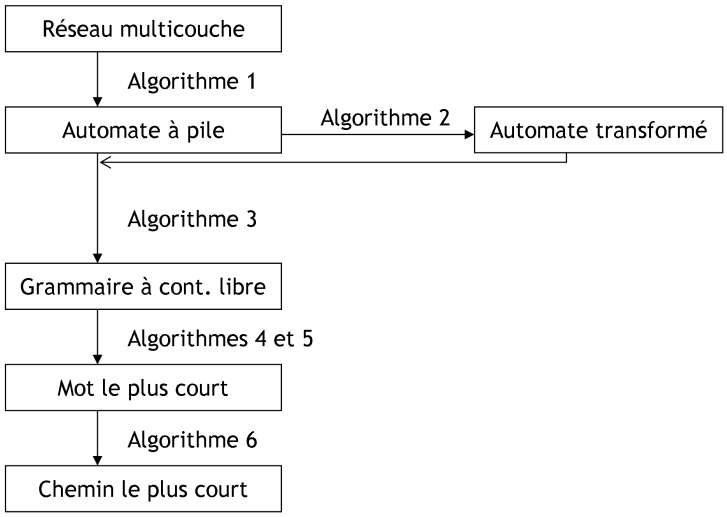


FIGURE 5.3 – Approche pour calculer le chemin le plus court dans un réseau multicouche.

Ce qui permet de caractériser un chemin faisable à chaque nœud, ce n'est pas uniquement l'ensemble des nœuds et des liens déjà parcourus, mais également la pile de protocoles. En effet, à chaque encapsulation, le protocole encapsulé est empilé et le protocole courant change. Cette pile est ce qui permet à chaque nœud de savoir si l'on peut désencapsuler ou non, en observant quel est le protocole en sommet de pile. Un automate à pile s'avère donc un outil naturel pour modéliser un réseau multicouche. En effet, les empilements modélisent naturellement les encapsulations et les dépilements modélisent les désencapsulations. Notre approche est donc de transformer un réseau multicouche en automate à pile. Nous nous fixons d'abord comme objectif de calculer le chemin le plus court selon deux métriques : le nombre de liens ou le nombre d'encapsulations (et donc de désencapsulations). Si nous optons pour la deuxième métrique, l'automate à pile doit être transformé pour court-circuiter toutes les séquences de transitions sans empilement ni dépilement. En effet, le fait de ne compter que les empilements et les dépilements correspond à ne compter que les encapsulations et les désencapsulations. Le mot le plus court accepté par l'automate transformé sera également le mot nécessitant le moins d'empilements pour être accepté par l'automate. Il nous faut ensuite générer le mot le plus court accepté par l'automate (ou l'automate transformé, selon le cas). Pour cela, l'automate est converti en grammaire selon un algorithme classique. Un autre algorithme nous permet de connaitre la longueur du mot le plus court généré par cette grammaire (qui correspond au mot le plus court accepté par l'automate). Une fois cette longueur connue, plusieurs algorithmes de générations de mots permettent de générer le mot le plus court via la grammaire. Ce

mot est en fait la séquence de protocoles correspondant au chemin le plus court (selon l'une ou l'autre des métriques). Il est facile de retrouver le chemin (ou un des chemins, s'il y en a plusieurs), qui lui correspond. Cette approche est représentée par le diagramme en figure 5.3. Tous les algorithmes utilisés sont polynomiaux. Notre approche est donc la première solution polynomiale pour le calcul de chemins dans un réseau multicouche.

Cette approche est généralisée selon n'importe quelle métrique additive de longueur de chemin. En effet, en associant un poids à chaque couple (lien, fonction d'adaptation), on peut représenter différentes métriques et donc calculer le chemin le plus court selon plusieurs critères (le délai, le coût énergétique, etc.). L'ajout de ce coût se traduit par la définition d'une fonction de poids sur les transitions de l'automate, qui est donc redéfini en automate à pile pondéré. Cet automate est converti en grammaire pondérée où un poids est associé à chaque règle de production. Le mot de poids minimum est ici le mot ayant l'arbre de dérivation de poids minimum (le poids de l'arbre est la somme des poids des règles de production utilisées dans l'arbre). Un algorithme de Knuth [66] permet de calculer cet arbre (et donc le mot qui y correspond). Ce mot, qui est une séquence de protocoles, permet de retrouver le chemin de poids minimum. Cette généralisation ne modifie que légèrement la plupart des algorithmes. Cette approche reste polynomiale.

Cette approche ne peut malheureusement pas être utilisée pour calculer le chemin le plus court sous contrainte de bande passante ou sous plusieurs contraintes de QoS. En effet, cette approche ne permet pas de savoir à priori combien de fois le chemin le plus court passera par un même lien. On ne peut donc savoir si ce lien a des paramètres de QoS suffisants avant de calculer le chemin en entier. Pour résoudre ce problème, nous avons opté pour une approche différente en nous basant sur l'algorithme SAMCRA. SAMCRA est principalement un algorithme de recherche de chemins en largeur. Sa spécificité étant de définir plusieurs concepts qui réduisent sensiblement l'espace des solutions à explorer, et permettent à l'algorithme d'avoir une complexité moyenne polynomiale (bien qu'exponentielle dans le pire des cas). Le plus important de ces concept est celui de la *non-dominance*, qui définit un ordre partiel sur les chemins et permet de n'explorer que les chemins non-dominés. Nous nous proposons d'étendre la notion de non-dominance à la prise en compte des contraintes de compatibilité. Cette extension permet de calculer des chemins faisables et respectant des contraintes de QoS.

5.3 Modèle d'un réseau multicouche

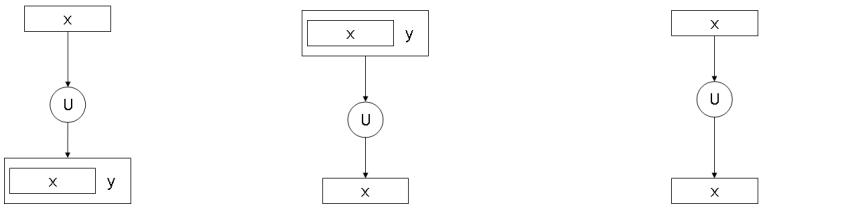
Un réseau multicouche avec des capacités d'encapsulation et de désencapsulation est modélisé par un un triplet $\mathcal{R} = (\mathcal{G},\mathcal{A},\wp)$ où :

- $\mathcal{G} = (\mathcal{V},\mathcal{E})$ est un graphe orienté qui modélise la topologie du réseau. Nous distinguons deux nœuds du graphe S et $D \in \mathcal{V}$ qui sont respectivement la source et la destination du chemin que nous voulons calculer ;
- $\mathcal{A} = \{a,b,c,\ldots\}$ est un ensemble fini de symboles que nous appellerons alphabet. Chaque symbole $x \in \mathcal{A}$ représente un protocole ;
- $\wp$ est l'ensemble des fonctions d'adaptation (encapsulations et désencapsulations) disponibles dans le réseau. $\wp = \{\bigcup_{U\in\mathcal{V}} \wp(U)\}$ où $\wp(U)$ est l'ensemble des fonctions d'adaptation disponibles sur le nœud U.

Nous définissons trois types de fonctions d'adaptation :

- Une encapsulation d'un PDU de x dans des PDU de y est notée (x,y). La figure 5.4(a) illustre une telle encapsulation effectuée par un nœud U. Si $(x,y) \in \wp(U)$, alors le nœud U est capable d'effectuer l'encapsulation (x,y) ;
- Une désencapsulation (extraction) de PDU de x depuis des PDU de y est notée $\overline{(x,y)}$. La figure 5.4(b) illustre une telle désencapsulation effectuée par le nœud U ;
- Une transmission passive (sans changement de protocole) de PDU de x est notée (x,x). La figure 5.4(c) illustre une telle transmission via un nœud U.

Remarque. La notation (x,x) pour désigner une transition passive n'est pas ambiguë. Elle ne peut pas être confondue avec une encapsulation de PDU de x dans d'autres PDU de x car une encapsulation d'un protocole dans lui-même n'est pas permise par l'architecture Pseudo-Wire. Néanmoins, si on travaille sur une autre architecture qui permet d'encapsuler un protocole dans lui-même (IPSec, par exemple),



(a) Encapsulation d'un protocole x dans un protocole y, $(x,y) \in \wp(U)$.

(b) Désencapsulation d'un protocole x d'un protocole y, $\overline{(x,y)} \in \wp(U)$.

(c) Transmission passive, $(x,x) \in \wp(U)$.

FIGURE 5.4 – Différentes fonctions d'adaptation de protocole sur un nœud U.

une modification de notation s'impose pour distinguer la transmission passive de PDU de protocole x et l'encapsulation de PDU de x dans des PDU de même type.

5.3.1 Notations et définitions

- Nous notons $\mathcal{ED}$ (resp. $\overline{\mathcal{ED}}$) l'ensemble de toutes les encapsulations (resp. désencapsulations) possibles, i.e., $\mathcal{ED} \subset \mathcal{A}^2$;
- Nous définissons $In(U) = \{x \in \mathcal{A}$ tel que $\exists y \in \mathcal{A}$ tel que (x,y) ou $\overline{(y,x)} \in \wp(U)\}$ comme étant l'ensemble des protocoles que le nœud U peut recevoir ;
- Nous définissons $Out(U) = \{y \in \mathcal{A}$ tel que $\exists x \in \mathcal{A}$ tel que (x,y) ou $\overline{(y,x)} \in \wp(U)\}$ comme étant l'ensemble des protocoles que le nœud U peut émettre ;
- Nous définissons $\mathcal{T}(U) = \{x \in \mathcal{A}$ *s.t.* $(x,x) \in \wp(U)\}$ comme étant l'ensemble des protocoles que le nœud U peut transmettre passivement ;
- Nous définissons $\overline{\mathcal{A}} = \{\overline{x} : x \in \mathcal{A}\}$ comme étant l'ensemble des symboles de l'alphabet $\mathcal{A}$ surlignés. Par souci de simplicité, nous considérerons qu'un symbole deux fois surligné est équivalent à un symbole non surligné, nous ne distinguerons pas les deux, i.e., $\forall x \in \mathcal{A}$, $\overline{\overline{x}} = x$.

Considérons un réseau $\mathcal{R} = (\mathcal{G},\mathcal{A},\wp)$, un nœud source $S \in \mathcal{V}$, un nœud destination $D \in \mathcal{V}$. Un chemin C est défini comme étant une séquence $C = S,x^1,U_1,x^2,\ldots,U_{n-1},x^n,D$ où chaque U_i est un nœud du réseau et où chaque $x^i \in \mathcal{A} \cup \overline{\mathcal{A}}$.

Nous définissons également :

- $T_C = x^1 \ldots x^n$ comme étant la séquence de protocoles du chemin C. La séquence T_C est appelée la **trace** de C. Pour chaque x^i :
  - Si $x^i = a$ and $x^{i+1} = b$ ou $\overline{b}$, cela signifie que U_i encapsule le protocole a dans b ;
  - Si $x^i = \overline{a}$ et $x^{i+1} = b$ ou $\overline{b}$, cela signifie que U_i désencapsule (ou extrait) le protocole b du protocole a ;
  - Si $x^i = a$ et $x^{i+1} = a$ ou $\overline{a}$, cela signifie que U_i peut transmettre passivement le protocole a.
- La **séquence de transition** de C, notée H_C, est la séquence $\beta_1,\ldots,\beta_n$ obtenue à partir de C comme suit :
  - Pour chaque $i \in \{1,\ldots,n-1\}$:
    - Si $x^i = a \in \mathcal{A}$ et $x^{i+1} = b \in \mathcal{A}$ ou $x^{i+1} = \overline{b} \in \overline{\mathcal{A}}$ alors $\beta_i = (a,b)$
    - Si $x^i = \overline{b} \in \overline{\mathcal{A}}$ et $x^{i+1} = a \in \mathcal{A}\backslash\{b\}$ ou $x^{i+1} = \overline{a} \in \overline{\mathcal{A}}\backslash\{\overline{b}\}$ alors $\beta_i = \overline{(a,b)}$

    Il est à noter que la paire (a,a), $a \in \mathcal{A}$ peut apparaître dans la séquence de transitions étant donné qu'elle représente une transition passive. Néanmoins, une paire $\overline{(a,a)}$, $a \in \mathcal{A}$ ne peut pas apparaître car nous considérons qu'un protocole a ne peut être encapsulé (ni désencapsulé) de lui-même.
- La **séquence parenthésée** de C, notée $M_C = \beta'_1,\ldots,\beta'_m$, est obtenue à partir de H_C en y supprimant les transitions passives ((x,x) pour tout $x \in \mathcal{A}$).

Exemple. Considérons le chemin $C = S,a,U,b,V,\overline{b},W,a,D$ dans le réseau illustré par la figure 5.5. La séquence de transitions correspondant à C est $H_C = (a,b),(b,b),\overline{(a,b)}$ et la trace de C est $T_C = ab\overline{b}a$. La séquence parenthésée de C est $M_C = (a,b),\overline{(a,b)}$.

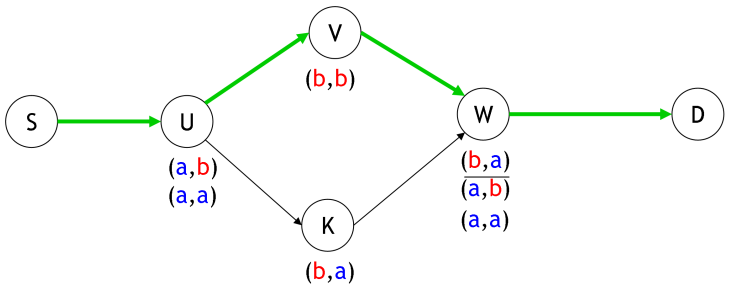


FIGURE 5.5 – Exemple d'un réseau multicouche. Les fonctions d'adaptation de chaque nœud sont indiquées sous celui-ci. Le chemin en vert est faisable si on emploie les fonctions d'adaptation appropriées.

5.4 Calcul de chemins sans contraintes de Qualité de Service

5.4.1 Définition formelle du problème

Notons par ε le mot vide (de 0 caractères) et notons par « $\bullet$ » l'opération de concaténation des mots. Soit C un chemin et H_C sa séquence de transitions. La définition suivante donne une caractérisation des chemins faisables :

Définition 1. *Une séquence M_C obtenue à partir de H_C est* valide *si et seulement si $M_C \in \mathcal{L}$, où $\mathcal{L}$ est le langage formel récursivement défini par :*

$$\mathcal{L} = \varepsilon \cup \left(\bigcup_{\substack{(x,y)\in\mathcal{A}^2 \\ x\neq y}} (x,y) \bullet \mathcal{L} \bullet \overline{(x,y)} \right) \bullet \mathcal{L}$$

Définition 2. *Un chemin C est* faisable *dans $\mathcal{R}$ de S à D si et seulement si :*
- *S,U_1,…,U_{n-1},D est un chemin dans $G = (\mathcal{V},\mathcal{E})$,*
- *Sa séquence parenthésée M_C est valide,*
- *Pour chaque $i \in \{1,\ldots,n-1\}$:*
  - *Si $x^i = a$ et $x^{i+1} = b$ ou $\overline{b}$ alors $\underline{(a,b)} \in P(U_i)$*
  - *Si $x^i = \overline{a}$ et $x^{i+1} = b$ ou $\overline{b}$ alors $\overline{(a,b)} \in P(U_i)$*
  - *Si $x^i = a$ et $x^{i+1} = a$ ou $\overline{a}$ alors $a \in \mathcal{T}(U_i)$*

Le langage $\mathcal{L}$ des séquences parenthésées valides est en fait un *langage de Dick généralisé* [71]. Ce langage est à contexte-libre mais il n'est pas régulier. De ce fait, les automates à piles sont naturellement adaptés à la modélisation de ce problème.

Exemple. Le réseau de la figure 5.5 a 6 nœuds et deux protocoles a et b. Les fonctions d'adaptation disponibles sur chaque nœud sont indiquées sous celui-ci. Par exemple, le nœud U peut encapsuler le protocole a dans le protocole b ($(a,b) \in \wp(U)$). Il peut également transmettre passivement le protocole a ($a \in \mathcal{T}(U)$). Dans ce réseau, le seul chemin faisable entre S et D est S,a,U,b,V,$\overline{b}$,W,a,D. Il nécessite l'encapsulation de a dans b par le nœud U, la transmission passive de b par le nœud V et la désencapsulation du protocole a à partir du protocole b par le nœud W. Ces opérations correspondent aux fonctions (a,b), (b,b) et $\overline{(a,b)}$ respectivement.

Définition du problème. Comme expliqué précédemment, notre but est de calculer le plus court chemin faisable entre les nœuds S et D. Nous nous donnons deux fonctions à minimiser, la longueur du chemin sera exprimée soit en nombre de liens, soit en nombre d'encapsulations et désencapsulations.

Formellement, le problème traité est le suivant :

$$\min_{C} \quad |H_C| \text{ ou } |M_C|$$
$$\text{tel que } C \text{ est faisable entre } S \text{ et } D.$$

5.4.2 Conversion d'un réseau multicouche en automate à pile

Dans cette section, nous nous intéressons à la transformation d'un réseau multicouche (tel que défini en section 5.3) en automate à pile. L'algorithme 3 prend en entrée un réseau $\mathcal{R} = (\mathcal{G},\mathcal{A},\wp)$, où $\mathcal{G} = (\mathcal{V},\mathcal{E})$, et le convertit en automate à pile $A_{\mathcal{R}} = (\mathcal{Q},\Sigma,\Gamma,\delta,S_A,z_0,\{D_A\})$ où $\mathcal{Q}$ est l'ensemble des états de l'automate, Σ est l'alphabet d'entrée, Γ est l'alphabet de la pile, δ est la fonction de transition, S_A est l'état initial, z_0 le symbole de fond de pile, D_A [35] est le seul état final et ε est le mot vide. Nous rappelons que Γ^* est la *fermeture de Kleene* (ou *fermeture itérative*) de l'ensemble Γ. La fermeture de Kleene d'un ensemble de symboles X est l'ensemble des mots sur X, y compris le mot vide ε.

L'automate $A_{\mathcal{R}}$ est obtenu à partir du réseau $\mathcal{R}$ comme suit :

- Un état U_x est créé dans l'automate pour chaque nœud $U \in \mathcal{V}$ et chaque symbole $x \in In(U)$, sauf pour le nœud source S pour lequel un seul état S_A est crée. Cet état sera l'état initial de l'automate ;
- Le symbole en sommet de pile correspondra au dernier protocole encapsulé ;
- Si l'automate est dans un état U_x, alors le protocole courant est x ;
- Les fonctions d'adaptation seront converties en transitions dans l'automate. Une transition $t \in \delta$ est notée $(U_x,\langle x,\alpha,\beta\rangle,V_y)$ où $U_x \in \mathcal{Q}$ est l'état de l'automate avant la transition, $x \in \Sigma$ est le caractère en entrée (ou en lecture), $\alpha \in \Gamma$ est le symbole qui sera dépilé du sommet de pile, $\beta \in \Gamma^*$ est la séquence de symboles empilés, et V_y est l'état de l'automate après la transition. Donc, si $\beta = \alpha$ alors t est une transition passive ; si $\beta = x\alpha$ alors t est un empilement de x ; enfin, si $\beta = \emptyset$ alors t est un dépilement ;
- Une transmission passive d'un protocole x par le nœud U est modélisée par une transition sans empilement ni dépilement entre l'état U_x et l'état suivant V_x. Elle est notée $(U_x,\langle x,\alpha,\alpha\rangle,V_x)$;
- Une encapsulation d'un protocole x dans un protocole y par un nœud U est modélisée par un empilement du symbole x lors de la transition entre l'état U_x et l'état suivant V_y. Cet empilement est noté $(U_x,\langle x,\alpha,x\alpha\rangle,V_y)$ [36] ;
- Une désencapsulation d'un protocole y depuis un protocole x par un nœud U est modélisée par un dépilement du symbole y. Ce dépilement est noté $(U_x,\langle\overline{x},y,\emptyset\rangle,V_y)$.

Algorithme 3 Conversion d'un réseau en automate à pile

Entrée : un réseau $\mathcal{R} = (\mathcal{G} = (\mathcal{V},\mathcal{E}),\mathcal{A},\wp)$, un nœud source S et un nœud destination D
Sortie : Un automate à pile $A_{\mathcal{R}} = (\mathcal{Q},\Sigma,\Gamma,\delta,S_A,z_0,\{D_A\})$
(1) $\Sigma \leftarrow \mathcal{A} \cup \overline{\mathcal{A}}$; $\Gamma \leftarrow \mathcal{A} \cup \{z_0\}$
(2) Créer un seul état S_A correspondant au nœud S
(3) Pour chaque nœud $U \neq S$ et chaque protocole $x \in In(U)$, créer un état U_x
(4) Pour chaque état U_x tel que $(S,U) \in \mathcal{E}$ et $x \in Out(S)$
    Créer une transition $(S_A,\langle\epsilon,z_0,z_0\rangle,U_x)$
(5) Pour chaque lien $(U,V) \in \mathcal{E}$ tel que $U \neq S$, chaque $(x,y) \in \wp(U)$ et chaque $\alpha \in \Gamma \setminus \{x\}$
  (5.1) Si $x \in \mathcal{T}(U) \cap In(V)$, créer une transition $(U_x,\langle x,\alpha,\alpha\rangle,V_x)$ */* transition passive */*
  (5.2) Si $x \neq y$ et $y \in In(V)$, créer une transition $(U_x,\langle x,\alpha,x\alpha\rangle,V_y)$ */* encapsulation */*
(6) Pour chaque lien $(U,V) \in \mathcal{E}$ tel que $U \neq S$ et pour chaque $\overline{(y,x)} \in \wp(U)$
  (6.1)Si $x \in In(V)$, créer une transition $(U_x,\langle\overline{x},y,\emptyset\rangle,V_y)$ */* désencapsulation */*
(7) Créer un état final fictif D_A.
(8) Pour chaque $x \in In(D)$ créer une transition $(D_x,\langle x,z_0,\emptyset\rangle,D_A)$

Complexité de l'algorithme 3. Chaque nœud U du réseau est transformé en $|In(U)|$ états, mis à part le nœud source S. Un état final fictif est ajouté. Le nombre d'états dans le pire des cas est donc $2 + (|\mathcal{V}| - 1) \times |\mathcal{A}|$ donc en $O(|\mathcal{V}| \times |\mathcal{A}|)$. La complexité temporelle de l'algorithme 3 est en $O(\max((|\mathcal{V}| \times |\mathcal{A}|),(|\mathcal{E}| \times ((|\mathcal{A}| \times |\mathcal{E}\mathcal{D}|) + |\overline{\mathcal{E}\mathcal{D}}|)))))$ dans le pire des cas. Nous supposons que le graphe représentant la topologie du réseau est connexe, donc $|\mathcal{E}| \geq |\mathcal{V}| - 1$. Vu que $\mathcal{E}\mathcal{D} \subseteq \mathcal{A}^2$, alors $|\mathcal{E}\mathcal{D}| \leq |\mathcal{A}|^2$

35. Il ne faut pas confondre l'état D_A, qui est l'état final de l'automate $A_{\mathcal{R}}$, avec l'état D_a, qui est l'état correspondant au nœud D indexé par le protocole a.

36. Il est à noter que même si $x = \overline{a} \in \overline{\mathcal{A}}$, la transition correspondante sera de la forme $(U_a,\langle\overline{a},\alpha,a\alpha\rangle,V_y)$. Les symboles appartenant à $\overline{\mathcal{A}}$ ne sont utilisés que dans l'alphabet d'entrée. Les symboles qui indexent les états et les symboles appartenant à l'alphabet de la pile appartiennent à $\mathcal{A}$ (sauf z_0).

et $|\overline{\mathcal{ED}}| < |\mathcal{A}|^2$. La complexité dans le pire des cas est donc bornée par $O(|\mathcal{A}|^3 \times |\mathcal{E}|)$, qui est également une borne pour le nombre de transitions.

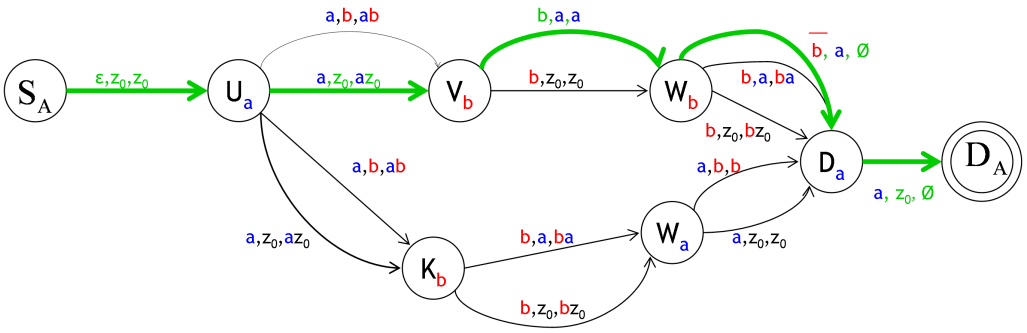


FIGURE 5.6 – L'automate à pile correspondant au réseau multicouche. Les transitions en gras correspondent au seul chemin faisable.

Exemple. L'algorithme 3 convertit le réseau multicouche sur la figure 5.5 en automate à pile illustré sur la figure 5.6. Par exemple, le lien (U,V) est converti en transitions $(U_a,\langle a,z_0,az_0\rangle,V_b)$ et $(U_a,\langle a,b,ab\rangle,V_b)$ (qui sont des empilements) car le nœud U est capable d'encapsuler le protocole a dans le protocole b (fonction d'adaptation (a,b)) avant de le transmettre au nœud V. Le nœud W est converti en deux états W_a et W_b car $In(W) = \{a,b\}$. Le lien (W,D) est converti en transitions $(W_b,\langle b,z_0,bz_0\rangle,D_a)$ et $(W_a,\langle b,a,ba\rangle,D_a)$ (empilements) en correspondance avec la fonction d'adaptation (a,b). Ce lien est aussi converti en transition $(W_b,\langle\overline{b},a,\emptyset\rangle,D_a)$ (dépilement) car le nœud W peut désencapsuler le protocole a depuis le protocole b (fonction d'adaptation $\overline{(a,b)}$) avant de le transmettre au nœud D. Le lien (V,W) est converti en transitions $(V_b,\langle b,z_0,z_0\rangle,W_b)$ et $(V_b,\langle b,a,a\rangle,W_b)$ (transitions passives) car le nœud V n'est capable que de transmettre passivement le protocole b (fonction (b,b)).

Proposition 3. *Soit $\mathcal{R} = (\mathcal{G} = (\mathcal{V},\mathcal{E}),\mathcal{A},\wp)$ un réseau, et soient $S \in \mathcal{V}$ et $D \in \mathcal{V}$ respectivement les nœuds source et destination. Le langage reconnu (ou accepté) par $A_\mathcal{R}$ est l'ensemble des traces des chemins faisables entre S et D.*

Démonstration. Soit $C = Sx^1U_1x^2 \dots x^iU_ix^{i+1} \dots U_{n-1}x^nD$ un chemin dans un réseau $\mathcal{R}$. Et soient sa trace $T_C = x^1 \dots x^ix^{i+1} \dots x^n$ et sa séquence parenthésée M_C. Nous nous proposons de démontrer que :

1. Si C est un chemin faisable, alors sa trace T_C est acceptée par $A_\mathcal{R}$;
2. Si un mot w est accepté par $A_\mathcal{R}$, alors il existe un chemin faisable C dans $\mathcal{R}$ dont la trace est le mot accepté $T_C = w$.

Soit donc C un chemin faisable dans $\mathcal{R}$ et soit sa trace T_C. Nous déduisons de T_C une séquence de transitions comme suit :

- Cette séquence commence par la transition $(S_A,\langle\epsilon,z_0,z_0\rangle,(U_1)_{x^1})$. Cette transition est dans l'automate car l'algorithme crée une transition $(S_A,\langle\epsilon,z_0,z_0\rangle,U_x)$ pour chaque état U_x tel que $(S,U) \in \mathcal{E}$ et $x \in Out(S)$, ce qui est le cas de U_{x^1}.
- Pour chaque lettre x^i dans T_C, nous considérons la transition suivante selon le cas :
  - Si $x^i \in \mathcal{A}$ alors :
    - si $x^{i+1} = x^i$ ou $x^{i+1} = \overline{x}^i$ (i.e., x^i et $\overline{x}^i$ sont le même symbole à un surlignement près), nous considérons la transition $((U_i)_{x^i},\langle x^i,\alpha,\alpha\rangle,(U_{i+1})_{x^i})$ où $\alpha \in \Gamma\backslash\{x^i\}$. Cette transition existe car une séquence x^ix^i ou $x^i\overline{x}^i$ dans la trace signifie qu'il y a une transition passive. La transition est créée dans la ligne (5.1) de l'algorithme 3.
    - si $x^{i+1} \neq x^i$ et $x^{i+1} \neq \overline{x}^i$ (i.e., x^i et $\overline{x}^i$ ne sont pas le même symbole), alors nous considérons la transition $((U_i)_{x^i},\langle x^i,\alpha,x^i\alpha\rangle,(U_{i+1})_{x^{i+1}})$, où $\alpha \in \Gamma\backslash\{x^i\}$. cette transition existe et est créée dans la ligne (5.2) de l'algorithme 3.
  - si $x^i \in \overline{\mathcal{A}}$ alors nous considérons la transition $((U_i)_{\overline{x}^i}, \langle x^i,x^{i+1},\emptyset\rangle, (U_{i+1})_{x^{i+1}})$. Cette transition existe et est créée dans la ligne (6.1) de l'algorithme 3.

Par souci de simplicité, nous considérons que si $i = n-1$ alors $U_{i+1} = D$. Il est clair que cette séquence de transitions dans $A_\mathcal{R}$ permet de reconnaître la trace T_C. Comme M_C est bien parenthésée, pour chaque

i tel que $x^i \in \overline{\mathcal{A}}$ alors le sommet de pile en atteignant l'état $(U_i)_{\overline{x}^i}$ contient $x^{i+1}\alpha$ (ou $\overline{x}^{i+1}\alpha$ en cas de plusieurs dépilements successifs). En atteignant l'état $(U_{i+1})_{x^{i+1}}$, le sommet de pile ne contient plus que α. Dans le cas où $i = n-1$, $\alpha = z_0$. La pile est donc vide en atteignant l'état D_A et T_C est accepté par l'automate.

Nous pouvons montrer de façon similaire que si un mot est accepté par l'automate, alors la séquence de transitions qui l'accepte correspond à un chemin faisable dans $\mathcal{R}$. □

5.4.3 Calcul du plus court chemin

Dans la section 5.4.2, nous avons décrit un algorithme qui transforme un réseau multicouche en automate à pile. L'étape suivante consiste à calculer un chemin qui minimise soit le nombre de liens, soit le nombre d'encapsulations. La méthode de calcul qui minimise le nombre de liens utilise directement l'automate à pile en sortie de l'algorithme 3, comme détaillé en section 5.4.3.1. Cependant, si nous voulons minimiser le nombre d'encapsulations, l'automate à pile doit être transformé dans le but de court-circuiter les séquences de transitions passives (i.e., sans empilement ni dépilement), comme détaillé dans la section 5.4.3.2. L'automate à pile original ou transformé (selon la fonction objectif choisie) est ensuite converti en grammaire à contexte libre. Le mot le plus court que génère cette grammaire correspond à la trace T_C du chemin le plus court (en nombre de liens ou en nombre d'encapsulations). Finalement, ayant la trace du chemin le plus court, il est aisé de retrouver le chemin lui-même, comme détaillé en section 5.4.5.

5.4.3.1 Minimiser le nombre de liens

Le nombre de symboles (ou lettres) d'un mot accepté par l'automate $A_\mathcal{R}$ est égal au nombre de liens dans le chemin faisable qui lui correspond (chaque symbole est un protocole utilisé sur un lien). L'étape de transformation de l'automate en section 5.4.3.2 doit être ignorée. L'automate $A_\mathcal{R}$ doit être directement converti en grammaire à contexte libre. Ensuite, le mot le plus court est généré par cette grammaire comme décrit en section 5.4.4.2. Le chemin faisable correspondant est calculé par l'algorithme 8 en page 93, comme détaillé en section 5.4.5.

5.4.3.2 Minimiser le nombre d'encapsulations

Pour ne compter que les encapsulations et désencapsulations dans un chemin faisable C, il ne faut compter que les empilements et dépilements dans la séquence de transitions qui accepte sa trace T_C. L'automate $A_\mathcal{R}$ doit être transformé en automate $A'_\mathcal{R}$ dans lequel toutes les séquences de transitions passives sont court-circuitées. Dans ce cas, la longueur du mot le plus court accepté par $A'_\mathcal{R}$ sera égale au nombre d'encapsulations et désencapsulations nécessaires pour l'accepter, plus une constante fixée.

Soit $\mathcal{Q}_a$ ($a \in \mathcal{A}$) l'ensemble des états de l'automate tels que $\mathcal{Q}_a = \{V_x \in \mathcal{Q} | x = a\}$, et soit $A^a_\mathcal{R}$ le *sous-automate* induit par $\mathcal{Q}_a$. Par analogie avec un sous-graphe, un sous-automate induit est un multigraphe avec des arcs marqués (qui sont en fait des transitions) tel que l'ensemble des sommets est $\mathcal{Q}_a$ et l'ensemble des arcs est l'ensemble des transitions entre les éléments de $\mathcal{Q}_a$. Comme il n'y a que des transitions passives entre les éléments de $\mathcal{Q}_a$, toutes les séquences de transitions dans un sous-automate sont passives [37]. Soit $P(U_x,V_x)$ la longueur du plus court chemin [38] entre U_x et V_x. Cette longueur peut-être calculée entre chaque paire d'éléments de $\mathcal{Q}_a$ grâce à l'algorithme de Floyd-Warshall [44, 114]. Soit $Succ(V_x)$ L'ensemble des états successeurs de V_x dans l'automate original $A_\mathcal{R}$, i.e., $Succ(V_x) = \{W_y \in \mathcal{Q} | \exists (V_x, \langle x,\alpha,\beta\rangle, W_y) \in \delta\}$.

L'algorithme 4 prend $A_\mathcal{R}$ en entrée et calcule l'automate transformé $A'_\mathcal{R} = (\mathcal{Q}', \Sigma', \Gamma', \delta', S_A, z_0, \{D_A\})$. L'automate $A'_\mathcal{R}$ est initialisé avec les valeurs de $A_\mathcal{R}$. Ensuite, l'algorithme 4 calcule le sous-automate $A^x_\mathcal{R}$ pour toute lettre de l'alphabet $x \in \mathcal{A}$ (étape (1)) ainsi que la distance $P(U_x,V_x)$ pour chaque paire d'états (U_x,V_x) dans le sous-automate (étape (2)). Chaque chemin entre une paire d'états de $A^x_\mathcal{R}$ est une séquence de transitions passives car le sous-automate ne contient que des transitions passives (par construction). Si un tel chemin existe (étape (3.1)), l'algorithme crée une nouvelle transition dans δ'. Cette nouvelle transition ira de l'état U_x vers chaque état dans $Succ(V_x)$ (étapes (3.1.2) et (2.1.3)). Les transitions ajoutées sont les mêmes que celles qui relient V_x à ces successeurs $W_y \in Succ(V_x)$, mais

37. Par abus de langage, nous appellerons *séquences de transitions passives* une séquence de transitions où il n'y a ni empilement ni dépilement.

38. Ici, *chemin* est à entendre dans le sens *séquence de transitions* dans l'automate.

avec une lettre en lecture indexée par le nombre de transitions passives entre U_x et V_x (c'est-à-dire la longueur $P(U_x,V_x)$) plus 1. Ces nouvelles transitions indiquent qu'il y avait dans l'automate original $A_{\mathcal{R}}$ une séquence de transitions qui correspond à la séquence de protocoles $xx\ldots x$ de longueur $P(U_x,V_x)+1$. Le symbole indexé est ajouté à l'alphabet Σ' du nouvel automate (étape (3.1.1)).

Algorithme 4 Transformation de l'automate $A_{\mathcal{R}}$ en $A'_{\mathcal{R}}$

Entrée : L'automate à pile $A_{\mathcal{R}} = (\mathcal{Q},\Sigma,\Gamma,\delta,S_A,z_0,\{D_A\})$
Sortie : L'automate transformé $A'_{\mathcal{R}} = (\mathcal{Q}',\Sigma',\Gamma',\delta',S_A,z_0,\{D_A\})$
$\mathcal{Q}' \leftarrow \mathcal{Q}$, $\Sigma' \leftarrow \Sigma$, $\Gamma' \leftarrow \Gamma$, $\delta' \leftarrow \delta$
Pour chaque $x \in \mathcal{A}$
(1) Calculer $A^x_{\mathcal{R}}$
(2) Calculer la longueur $P(U_x,V_x)$ entre chaque paire d'états U_x et V_x dans $A^x_{\mathcal{R}}$
(3) Pour chaque $U_x \in \mathcal{Q}_x$ et chaque $V_x \in \mathcal{Q}_x$ faire
(3.1) Si $P(U_x,V_x) < \infty$ */* *Il y a un chemin entre* U_x *et* V_x */*
(3.1.1) Ajouter le symbole $x_{P(U_x,V_x)+1}$ et $\overline{x}_{P(U_x,V_x)+1}$ à Σ
(3.1.2) Pour chaque $W_y \in Succ(V_x)\backslash\{U_x\}$ et chaque $(V_x,\langle x,\alpha,\beta\rangle,W_y) \in \delta$
Ajouter la transition $(U_x,\langle x_{P(U_x,V_x)+1},\alpha,\beta\rangle,W_y)$ à δ'
(3.1.3) Pour chaque $W_y \in Succ(V_x)\backslash\{U_x\}$ et chaque $(V_x,\langle \overline{x},\alpha,\beta\rangle,W_y) \in \delta$
Ajouter la transition $(U_x,\langle \overline{x}_{P(U_x,V_x)+1},\alpha,\beta\rangle,W_y)$ to δ'

Complexité de l'algorithme 4. Le calcul du sous-automate (étape (1)) se fait en $O(|\delta| + |\mathcal{Q}|)$, car il suffit de parcourir l'ensemble des nœuds et des transitions de l'automate et de ne garder que les états indexés par la lettre x et les transitions entre ces états. Calculer le chemin le plus court entre chaque paire de nœuds se fait grâce à l'algorithme de Floyd-Warshall en $O(|\mathcal{Q}_x|^3)$. Si $x = y$, il a au maximum $|\mathcal{A}| - 1$ transitions entre V_x et W_y dans $\mathcal{Q}$ (des transitions de la forme $(U_x,\langle x,\alpha,\alpha\rangle,W_x)$ pour toutes les valeurs possibles de α, excepté x, puisque la construction de l'automate interdit l'encapsulation d'un protocole x dans lui-même car l'architecture Pseudo-Wire ne le permet pas – il est donc impossible d'avoir x comme protocole courant et en même temps comme sommet de pile). Si $x \neq y$, il y a au plus $|\mathcal{A}|$ transitions entre V_x et W_y dans $\mathcal{Q}$ (le dépilement $(V_x,\langle \overline{x},y,\emptyset\rangle,W_y)$ et les empilements $(V_x,\langle x,\alpha,x\alpha\rangle,W_y)$ pour toutes les valeurs de $\alpha \in \Gamma$, excepté x). Et comme $|Succ(V_x)| < |\mathcal{Q}|$, la complexité des étapes (3.1.2) et (3.1.3) est bornée par $O(|\mathcal{Q}| \times |\mathcal{A}|)$. Cependant un état ne peut appartenir qu'à un seul sous-automate, la complexité de l'algorithme 4 est bornée par $O\left(\sum_{x\in\mathcal{A}} \left[|\delta| + |\mathcal{Q}| + |\mathcal{Q}_x|^3 + (|\mathcal{Q}_x|^2 \times |\mathcal{Q}| \times |\mathcal{A}|)\right]\right)$. De plus $\forall x \in \mathcal{A}, |\mathcal{Q}_x| \leq |\mathcal{V}|$ (voir l'algorithme 3 pour la construction de l'ensemble d'états $\mathcal{Q}$). En conséquence, la complexité de l'algorithme 4 est en $O\left(\max\left(|\mathcal{A}| \times (|\delta| + |\mathcal{Q}|), |\mathcal{A}| \times |\mathcal{V}|^3, |\mathcal{A}|^2 \times |\mathcal{V}|^2 \times |\mathcal{Q}|\right)\right)$, ce qui correspond à $O\left(\max(|\mathcal{A}|^4 \times |\mathcal{E}|, |\mathcal{A}|^3 \times |\mathcal{V}|^3)\right)$ dans le modèle de réseau multicouche. Le nombre de transitions dans δ' est borné par $|\delta| + O\left(\sum_{x\in\mathcal{A}}(|\mathcal{Q}_x|^2 \times |\mathcal{Q}| \times |\mathcal{A}|)\right)$, ce qui correspond à $O(|\mathcal{A}|^3 \times |\mathcal{V}|^3)$ dans le modèle de réseau multicouche.

Exemple. L'algorithme 4 transforme l'automate illustré par la figure 5.6 en l'automate illustré par la figure 5.7. Par exemple, en considérant le protocole b, l'algorithme calcule le sous-automate induit par les états V_b, W_b et K_b. La longueur $P(V_b,W_b) = 1$ est ensuite calculée. Pour court-circuiter la séquence de transitions $(V_b,\langle b,a,a\rangle,W_b)$ $(W_b,\langle \overline{b},a,\emptyset\rangle,D_a)$, la transition $(V_b,\langle \overline{b_2},a,\emptyset\rangle,D_a)$ est créée et ajoutée à l'automate transformé.

Soit $L(A_{\mathcal{R}})$ le langage (i.e., l'ensemble de mots) accepté par $A_{\mathcal{R}}$, et soit $L(A'_{\mathcal{R}})$ le langage accepté par $A'_{\mathcal{R}}$. Enfin, soit $f : \Sigma' \rightarrow \Sigma^*$ la fonction définie par :

– Si $x_i = a_i \in \mathcal{A}'$ alors $f(x_i) = \underbrace{aa\ldots aa}_{i \text{ fois}}$
– Si $x_i = \overline{a_i} \in \overline{\mathcal{A}'}$ alors $f(x_i) = \underbrace{aa\ldots a\overline{a}}_{i \text{ fois}}$

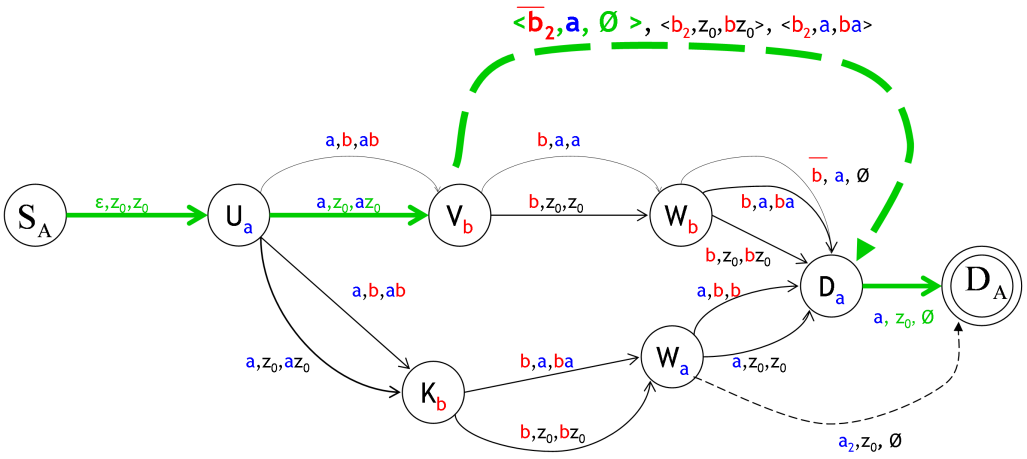


FIGURE 5.7 – L'automate à pile transformé correspondant à l'automate sur la figure 5.6.

Puis, par extension du domaine de f à $(\Sigma')^*$:

$$f : (\Sigma')^* \to \Sigma^*$$
$$w' = x_i^1 x_j^2 \dots x_k^n \to f(w') = f(x_i^1) f(x_j^2) \dots f(x_k^n)$$

Par souci de simplicité, nous ne ferons pas de distinction entre x et x_1 pour tout $x \in \mathcal{A}$. Nous noterons $f(L(A'_{\mathcal{R}}))$ le langage (ou l'ensemble des mots) accepté par $A'_{\mathcal{R}}$ transformé par f (i.e. $f(L(A'_{\mathcal{R}})) = \{f(w')$ tel que $w' \in L(A'_{\mathcal{R}})\}$).

Il est évident que f n'est pas une bijection puisque $f(x_i x_j) = f(x_{i+j})$. Pour opérer la transformation entre $L(A_{\mathcal{R}})$ et $L(A'_{\mathcal{R}})$, Nous définissons $g : \Sigma^* \to (\Sigma')^*$ telle que :

– Pour chaque $w = \underbrace{x x \dots x}_{i \text{ fois}} \underbrace{y y \dots y}_{j \text{ fois}} \dots \underbrace{z z \dots z}_{k \text{ fois}} \in \Sigma^*$, $g(w) = x_i y_j \dots z_k$.

En d'autres termes : $w' = g(w)$ est le mot le plus court dans $(\Sigma')^*$ tel que $f(w') = w$.

Les lemmes et les propositions suivantes montrent que le langage accepté par l'automate transformé $A'_{\mathcal{R}}$ est « *équivalent* » au langage accepté par l'automate original $A_{\mathcal{R}}$. Ici, *équivalent* signifie que chaque mot accepté par $A'_{\mathcal{R}}$ peut être associé à un mot accepté par $A_{\mathcal{R}}$, et ceci via la fonction f. De plus, l'automate transformé a une nouvelle propriété : **il y a une relation linéaire entre la longueur d'un mot accepté et le nombre minimum d'encapsulations et de désencapsulations nécessaires pour l'accepter.** Cette propriété permet de trouver le mot nécessitant le nombre minimum d'encapsulations et de désencapsulations accepté par l'automate original $A_{\mathcal{R}}$. Cela peut-être fait simplement en calculant le mot le plus court accepté par l'automate transformé $A'_{\mathcal{R}}$.

Lemme 1. *Le langage $f(L(A'_{\mathcal{R}}))$ accepté par $A'_{\mathcal{R}}$ puis transformé par f, est le même que le langage $L(A_{\mathcal{R}})$ accepté par $A_{\mathcal{R}}$.*

Démonstration. Nous démontrons cette égalité par double inclusion :

1. $L(A_{\mathcal{R}}) \subseteq f(L(A'_{\mathcal{R}}))$: $\forall w \in L(A_{\mathcal{R}}), f(w) = w$ (nous rappelons que $x = x_1$). Donc $f(L(A_{\mathcal{R}})) = L(A_{\mathcal{R}})$ (1). D'un autre côté, la construction de $A'_{\mathcal{R}}$ par l'algorithme 4 ne supprime aucun symbole de l'alphabet, aucune transition, aucun état (que cet état soit final ou non) de $A_{\mathcal{R}}$. Donc $L(A_{\mathcal{R}}) \subseteq L(A'_{\mathcal{R}})$. Et donc $f(L(A_{\mathcal{R}})) \subseteq f(L(A'_{\mathcal{R}}))$ (2) .
   De (1) et (2) Nous déduisons $L(A_{\mathcal{R}}) \subseteq f(L(A'_{\mathcal{R}}))$.
2. $L(A_{\mathcal{R}}) \supseteq f(L(A'_{\mathcal{R}}))$: Soit $w' = x_i^1 x_j^2 \dots x_l^n$ un mot dans $L(A'_{\mathcal{R}})$ et soit $t'_1 t'_2 \dots t'_{n+1}$ une séquence de transitions acceptant w'. Pour chaque transition $t'_m = (U_{x^{m-1}}, \langle x_k^{m-1}, \alpha, \beta \rangle, W_{x^m})$ $(1 < m < n)$ de cette séquence, telle que $t'_m \in \delta' \backslash \delta$, il y a une séquence de $k-1$ transitions passives dans $A_{\mathcal{R}}$ (puisque la construction de t'_m dans $A'_{\mathcal{R}}$ nécessite l'existence d'une telle séquence dans $A_{\mathcal{R}}$). Cette séquence débute par l'état $U_{x^{m-1}}$ et est suivie par la transition $(V_{x^{m-1}}, \langle x^{m-1}, \alpha, \beta \rangle, W_{x^m})$. Donc, cette séquence dans $A_{\mathcal{R}}$ correspond à $f(x_k^{m-1})$. Et pour chaque t'_m dans $A'_{\mathcal{R}}$, ou bien $t'_m \in \delta$ et donc

t'_m correspond à x_k^{m+1} (si $k = 1$), ou bien $t'_m \in \delta' \backslash \delta$ et donc il y a une séquence de transitions dans $A_\mathcal{R}$ qui correspond à $f(x_k^{m+1})$ (si $k > 1$). Et comme, par définition, $f(w') = f(x_i^1)f(x_j^2)\dots f(x_l^n)$, $f(w') \in L(A_\mathcal{R})$. Ainsi $f(L(A'_\mathcal{R})) \subseteq L(A_\mathcal{R})$.

□

Soit w' le mot le plus court accepté par $A'_\mathcal{R}$. Soit $Nb_{push}^{A'_\mathcal{R}}(w')$ le nombre minimum d'empilements dans une séquence de transitions acceptant w' dans $A'_\mathcal{R}$. Et soit $Nb_{pop}^{A'_\mathcal{R}}(w')$ le nombre minimum de dépilements dans une séquence de transitions acceptant w' dans $A'_\mathcal{R}$.

Lemme 2. *La longueur de la plus courte séquence de transitions acceptant w' dans $A'_\mathcal{R}$ est égale à $1 + Nb_{push}^{A'_\mathcal{R}}(w') + Nb_{pop}^{A'_\mathcal{R}}(w')$. De plus, $Nb_{pop}^{A'_\mathcal{R}}(w') = Nb_{push}^{A'_\mathcal{R}}(w') + 1$.*

Démonstration. Tout d'abord, nous prouvons que w' ne peut pas être de la forme $\dots x_i^m x_j^{m+1} \dots$ avec $x^m = x^{m+1}$. Soit U_{x^m} l'état dans lequel $A'_\mathcal{R}$ est avant de lire le symbole x_i^m. Soit $V_{x^{m+1}}$ l'état dans lequel est $A'_\mathcal{R}$ après la lecture du symbole x_i^m et avant la lecture du symbole x_j^{m+1}. Enfin soit W_y l'état dans lequel est $A'_\mathcal{R}$ après la lecture du symbole x_j^{m+1}.

Si $x^m = x^{m+1}$, alors, par construction, il y a une transition $(U_{x^m},\langle x_{i+j}^m,\alpha,\beta\rangle,W_y)$ (où α,β est un dépilement de y ou un empilement de x^m if $x^m \neq y$). Alors, si nous remplaçons $x_i^m x_j^{m+1}$ par x_{i+j}^m dans w', le mot obtenu serait plus court que w' et il serait aussi accepté par $A'_\mathcal{R}$. Donc le mot le plus court accepté par $A'_\mathcal{R}$ est $w' = x_i^1 x_j^2 \dots x_k^n$ où $x^m \neq x^{m+1}$ avec $(1 \leq m < n)$.

Par construction de $A'_\mathcal{R}$, pour chaque symbole x_i^m dans le mot w', il y a une transition d'empilement $(U_{x^m},\langle x_i^m,\alpha,x^m\alpha\rangle,W_y)$ si $x_i^m \in \mathcal{A}'$ ou une transition $(U_{x^m},\langle x_i^m,y,\emptyset\rangle,W_y)$ si $x_i^m \in \overline{\mathcal{A}'}$. Donc toutes les transitions dans la plus courte séquence de transitions acceptant w' sont des empilements ou des dépilements, excepté la première transition à partir de l'état initial $(S_A,\langle\epsilon,z_0,z_0\rangle,U_{x^1})$ et le dépilement final $(V_{x^n},\langle x_k^n,z_0,\emptyset\rangle,D_A)$. Le nombre des autres dépilements est égal au nombre d'empilements (ce qui est nécessaire pour avoir z_0 au sommet de la pile avant la transition finale). □

Maintenant, soit w un mot accepté par $A_\mathcal{R}$, soit $Nb_{push}^{A_\mathcal{R}}(w)$ le nombre minimum d'empilements dans une séquence de transitions acceptant w dans $A_\mathcal{R}$, et soit $Nb_{pop}^{A_\mathcal{R}}(w)$ le nombre minimum de dépilements dans une séquence de transitions acceptant w dans $A_\mathcal{R}$.

Lemme 3. *Pour chaque mot $w \in L(A_\mathcal{R})$:*

- $Nb_{pop}^{A_\mathcal{R}}(w) = Nb_{pop}^{A'_\mathcal{R}}(g(w))$
- $Nb_{push}^{A_\mathcal{R}}(w) = Nb_{push}^{A'_\mathcal{R}}(g(w))$

Démonstration. Soit $t_1 t_2 \dots t_n$ la séquence de transitions acceptant w dans $A_\mathcal{R}$ avec un nombre minimum d'empilements et de dépilements. Pour chaque séquence $t_i t_{i+1} \dots t_j$ de transitions passives suivie par un empilement ou un dépilement, il y a une transition avec le même empilement ou dépilement et le même symbole en lecture indexé par $j - i + 1$. Alors $g(w) \in L(A'_\mathcal{R})$.

Soit $t'_1 t'_2 \dots t'_{n'}$ la séquence de transitions ayant le minimum d'empilements et dépilements et acceptant $g(w)$ dans $A'_\mathcal{R}$. Il est évident que $f(g(w)) = w$. Ainsi, s'il y a une séquence $t''_1 t''_2 \dots t''_{n'}$ qui accepte $g(w)$ avec moins d'empilements et de dépilements que $t'_1 t'_2 \dots t'_{n'}$, alors chaque $t''_i \in \delta' \setminus \delta$ peut être remplacée par une séquence de transitions passives suivies par un empilement ou un dépilement dans $A_\mathcal{R}$. Et cette séquence accepte $f(g(w))$ (qui est égal à w) avec moins d'empilements et de dépilements que dans la séquence $t_1 t_2 \dots t_n$, ce qui contredit le fait que cette séquence minimise le nombre d'empilements et de dépilements pour accepter w. □

Proposition 4. *Le mot accepté par $A_\mathcal{R}$ qui minimise le nombre d'empilements et de dépilements est $f(w')$, où w' est le mot le plus court (i.e., ayant le minimum de symboles) accepté par $A'_\mathcal{R}$.*

Démonstration. Par le lemme 2, le mot le plus court accepté par $A'_\mathcal{R}$ minimise le nombre d'empilements et de dépilements dans $A'_\mathcal{R}$ pour être accepté (puisque le nombre d'empilements et de dépilements croît linéairement en fonction de la longueur du mot). Soit w' ce mot.

Supposons qu'il y a un mot w accepté par $A_\mathcal{R}$ tel que $Nb_{push}^{A_\mathcal{R}}(w) < Nb_{push}^{A'_\mathcal{R}}(w')$. Par le lemme 3, $Nb_{push}^{A'_\mathcal{R}}(g(w)) = Nb_{push}^{A_\mathcal{R}}(w)$, et donc $Nb_{push}^{A'_\mathcal{R}}(g(w)) < Nb_{push}^{A'_\mathcal{R}}(w')$, puisque $w' = \arg\min Nb_{push}^{A'_\mathcal{R}}$. Par l'absurde, $f(w')$ est le mot qui minimise le nombre d'empilements et de dépilements dans $A_\mathcal{R}$ pour être accepté. □

5.4.4 Génération du mot le plus court

Pour calculer le mot le plus court accepté par $A_{\mathcal{R}}$ (resp. $A'_{\mathcal{R}}$), la grammaire à contexte libre $G_{\mathcal{R}}$ telle que $L(G_{\mathcal{R}}) = L(A_{\mathcal{R}})$ (resp. $L(A'_{\mathcal{R}})$) est calculée. Cela nous permettra de générer la séquence de protocoles (ou trace) la plus courte grâce à des algorithmes existants. Le mot le plus court de $L(G_{\mathcal{R}})$ doit être généré par cette grammaire.

5.4.4.1 Conversion de l'automate à pile en grammaire à contexte libre

La conversion d'un automate à pile en grammaire à contexte libre est une opération classique en Théorie des Langages. Nous adaptons une méthode générale décrite dans [58] pour convertir $A_{\mathcal{R}}$(resp. $A'_{\mathcal{R}}$) en grammaire. La méthode décrite dans [58] permet de transformer chaque transition d'automate à pile en un ensemble de règles de production d'une grammaire à contexte libre. Les transitions considérées peuvent empiler plusieurs symboles à la fois, par exemple $(U_x,\langle x,\alpha,\beta_n \dots \beta_2\beta_1\alpha\rangle,V_y)$ qui empile tous les symboles $\beta_n,\dots,\beta_2,\beta_1$ à la fois. Les automates à pile que nous considérons, que ce soit $A_{\mathcal{R}}$ ou $A'_{\mathcal{R}}$, ne possèdent pas ce genre de transitions. Nous ne traitons que les dépilements, transitions passives et empilements d'un seul symbole (transitions de la forme $(U_x,\langle x,\alpha,x\alpha\rangle,V_y)$). L'adaptation que nous proposons ne traite que ces cas et est donc plus simple que la méthode générale de [58].

L'algorithme 5 prend un automate à pile en entrée et produit une grammaire à contexte libre $G_{\mathcal{R}} = (\mathcal{N},\Sigma,[S_G],\mathcal{P})$ (resp. $(\mathcal{N},\Sigma',[S_G],\mathcal{P})$) où :

- $\mathcal{N}$ est l'ensemble des non terminaux (variables),
- Σ (resp. Σ') est l'alphabet en entrée,
- $[S_G]$ est le symbole initial (non terminal initial, ou axiome) ,
- $\mathcal{P}$ est l'ensemble des règle de production.

Excepté $[S_G]$, tous les non terminaux sont de la forme $[UXV]$ où $U,V \in \mathcal{Q}$ et $X \in \Gamma$ (resp. $\mathcal{Q}'$ et Γ'). La preuve de correction de cette méthode est décrite dans [58].

Algorithme 5 Conversion de l'automate à pile en grammaire à contexte libre

Entrée : Automate à pile $A_{\mathcal{R}} = (\mathcal{Q},\Sigma,\Gamma,\delta,S_A,z_0,\{D_A\})$ (resp. l'automate transformé $A'_{\mathcal{R}} = (\mathcal{Q}',\Sigma',\Gamma',\delta',S_A,z_0,\{D_A\})$)
Sortie : Grammaire à contexte libre $G_{\mathcal{R}} = (\mathcal{N},\Sigma,[S_G],\mathcal{P})$ (resp. $(\mathcal{N},\Sigma',[S_G],\mathcal{P}')$)
(1) Créer le non terminal $[S_G]$
(2) Pour chaque état $U_x \in \mathcal{Q}$
  (2.1) Créer le non terminal $[S_A z_0 U_x]$ et la production $[S_G] \rightarrow [S_A z_0 U_x]$
(3) Pour chaque transition $(U_x,\langle x,\alpha,\beta\rangle,V_y)$
  (3.1) Si $\beta = \emptyset$ (dépilement), créer le non terminal $[U_x\alpha V_y]$ et la production $[U_x\alpha V_y] \rightarrow x$
  (3.2) Si $\beta = \alpha$ (transition passive), créer pour chaque état $W \in \mathcal{Q}$ (resp. $\mathcal{Q}'$)
    (3.2.1) Les non terminaux $[U_x\alpha W]$ et $[V_y\alpha W]$
    (3.2.2) La production $[U_x\alpha W] \rightarrow x[V_y\alpha W]$
  (3.3) Si $\beta = x\alpha$, $x \in \Gamma$ (empilement), créer pour chaque couple d'états $(W,W') \in \mathcal{Q}^2$ (resp. $\mathcal{Q}'^2$)
    (3.3.1) Les non terminaux $[U_x\alpha W']$, $[V_y\alpha W]$ et $[W\alpha W']$
    (3.3.2) La production $[U_x\alpha W'] \rightarrow x[V_y x W][W\alpha W']$

Complexité de l'algorithme 5. Le nombre de règles de production dans $G_{\mathcal{R}}$ est borné par $1+|\mathcal{Q}|+(|\delta|\times|\mathcal{Q}|^2)$ (resp. $1+|\mathcal{Q}'|+(|\delta'|\times|\mathcal{Q}'|^2)$. Vu que les non terminaux (excepté $[S_G]$) sont de la forme $[U_x\alpha V_y]$ avec $U_x,V_y \in \mathcal{Q}$ et $\alpha \in \Gamma$, le nombre de non terminaux est borné par $O(|\mathcal{A}|\times|\mathcal{Q}|^2)$ (resp. $O(|\mathcal{A}|\times|\mathcal{Q}|^2)$). La complexité en temps de l'algorithme 5 est en $O(|\delta|\times|\mathcal{Q}|^2)$ (resp. $O(|\delta'|\times|\mathcal{Q}'|^2)$) dans le pire des cas, ce qui correspond à $O(|\mathcal{A}|^5\times|\mathcal{V}|^2\times|\mathcal{E}|)$ (resp. $O(|\mathcal{A}|^5\times|\mathcal{V}|^5)$) dans le modèle de réseau multicouche.

Exemple. Cet algorithme convertit l'automate illustré par la figure 5.7 en grammaire à contexte libre. La figure 5.8 montre un sous-ensemble des règles de production de cette grammaire. Ce sous-ensemble permet de générer la plus courte trace d'un chemin faisable entre les nœuds S et D dans le réseau illustré par la figure 5.5. Par exemple, la transition $(V_b,\langle\overline{b}_2,a,\emptyset\rangle,D_a)$ donne la règle de production $[V_b a D_a] \rightarrow \overline{b}_2$. La transition $(U_a,\langle a,z_0,az_0\rangle,V_b)$ donne les règles de production $[U_a z_0 X'] \rightarrow a[V_b a X][X z_0 X']$ où $X,X' \in \mathcal{Q}'$,

(1) $[S_G] \rightarrow [S_A z_0 D_A]$
(2) $[S_A z_0 D_A] \rightarrow \epsilon[U_a z_0 D_A]$
(3) $[U_a z_0 D_A] \rightarrow a[V_b a D_a][D_a z_0 D_A]$
(4) $[V_b a D_a] \rightarrow \bar{b}_2$
(5) $[D_a z_0 D_A] \rightarrow a$

FIGURE 5.8 – Sous-ensemble de règles de production de la grammaire à contexte libre.

y compris la règle de production $[U_a z_0 D_A] \rightarrow a[V_b a D_a][D_a z_0 D_A]$.

Remarque. Il y a deux mécanismes d'acceptation de mots définis sur les automates à pile : un mot en entrée est accepté soit par *pile vide* (si la pile est vide après la lecture du mot, – c'est-à-dire que le symbole de fond de pile z_0 est dépilé), soit par *état final* (si un état final est atteint après la lecture du mot,– même si la pile n'est pas vide). La méthode définie dans [58] (et que nous avons adaptée) fonctionne sur les automates qui acceptent les mots par pile vide. Dans notre cas, cette restriction ne pose pas de problème étant donné que, par construction, $A_{\mathcal{R}}$ et $A'_{\mathcal{R}}$ acceptent les mots par pile vide **et** état final, puisque les seules transitions qui permettent d'accéder à l'état final (unique) sont les transitions qui vident la pile (les transitions de la forme $(D_x,\langle x,z_0,\emptyset\rangle,D_A)$, $x \in In(D)$). De ce fait, l'algorithme 5 fonctionne correctement avec $A_{\mathcal{R}}$ ou $A'_{\mathcal{R}}$ en entrée.

5.4.4.2 Le mot le plus court généré par une grammaire à contexte libre

Longueur du mot le plus court. Pour calculer le mot le plus court généré par $G_{\mathcal{R}}$, la fonction ℓ associe à chaque non terminal la longueur du mot le plus court qui peut être obtenu en partant de ce non terminal. De ce fait, $\ell([S_G])$ est la longueur du mot le plus court généré par $G_{\mathcal{R}}$. L'algorithme 6 a été proposé par Y. Filmus et K. Ghasemloo [60] en réponse à une question sur le mot le plus court accepté par un automate à pile [69], sur le forum `http://cstheory.stackexchange.com`. Il s'est avéré plus tard que l'algorithme 6 est un cas particulier d'un algorithme plus général de Knuth [66] qui calcule des valeurs associées aux non terminaux d'une grammaire à contexte libre supérieure (ces valeurs peuvent être la longueur du mot le plus court généré par chaque non terminal – ce qui est le cas qui nous intéresse, mais ces valeurs peuvent être également la profondeur minimum d'un arbre de dérivation à partir de chaque non terminal, ou des booléens indiquant si le langage généré par chaque non terminal est vide ou non, etc. Voir [66] pour les différentes applications).

Plus formellement, $\ell : \{\mathcal{N} \cup \Sigma \cup \{\epsilon\}\}^*$ou $\{\mathcal{N} \cup \Sigma' \cup \{\epsilon\}\}^* \rightarrow \mathbb{N} \cup \{\infty\}$ telle que :
- Si $w = \epsilon$ alors $\ell(w) = 0$,
- Si $w \in \Sigma$ ou Σ' alors $\ell(w) = 1$,
- Si $w = \alpha_1 \dots \alpha_n$ (où $\alpha_i \in \{\mathcal{N} \cup \Sigma \cup \{\epsilon\}\}$ ou $\{\mathcal{N} \cup \Sigma' \cup \{\epsilon\}\}$) alors $\ell(w) = \sum_{i=1}^{n} \ell(\alpha_i)$.

L'algorithme 6 calcule la valeur de $\ell([X])$ pour chaque non terminal $[X] \in \mathcal{N}$

Algorithme 6 Calcul de la valeur de $\ell([X])$ pour chaque non terminal $[X] \in \mathcal{N}$

Entrée : $G_{\mathcal{R}} = (\mathcal{N},\Sigma,[S_G],\mathcal{P})$ ou $(\mathcal{N},\Sigma',[S_G],\mathcal{P})$
Sortie : $\ell([X])$ pour chaque non terminal $[X]$
(1) Initialiser chaque $\ell([X])$ à ∞
(2) Tant qu'il y a au moins une valeur $\ell([X])$ mise à jour lors de l'itération précédente
  (2.1) Pour chaque règle de production $[X] \rightarrow \alpha_1 \dots \alpha_n$ dans $\mathcal{P}$
    (2.1.1) $\ell([X]) \leftarrow \min\{\ell([X]), \sum_{i=1}^{n} \ell(\alpha_i)\}$

Proposition 5. *L'algorithme 6 se termine après $|\mathcal{N}|$ itérations dans le pire des cas, et chaque $\ell([X])$ ($[X] \in \mathcal{N}$) obtenu à la fin est la longueur (en nombre de symboles) du mot le plus court généré à partir du non terminal $[X]$.*

Démonstration. Nous prouvons qu'à chaque itération (exceptée la dernière), il y a au moins un non terminal $[X]$ tel que $\ell([X])$ est mis à jour avec la longueur du mot le plus court que $[X]$ génère. Donc, la mise à jour de toutes les valeurs de $\ell([X])$ est faite au pire après $|\mathcal{N}|$ itérations. Nous procédons par

récurrence sur le nombre d'itérations :

Cas trivial : Il n'y a pas d'ε-production [39] dans $G_{\mathcal{R}}$, et il y a au moins une production de la forme $[X] \to x$ où $[X] \in \mathcal{N}$ et $x \in \Sigma$ (resp. Σ') (voir l'algorithme 5 pour la construction de $G_{\mathcal{R}}$). Pour chaque $[X] \in \mathcal{N}$ tel que $\{[X] \to x\} \in \mathcal{P}$,$\ell([X]) = 1$. L'algorithme 6 assigne ces valeurs à chaque $\ell([X])$ lors de la première itération.

Cas général : Supposons qu'après l'itération n, il y a au moins n' ($n \leq n' < |\mathcal{N}|$) non terminaux $[X]$ tels que l'algorithme a calculé la valeur correcte de $\ell([X])$. Il reste donc $|\mathcal{N}| - n'$ non terminaux $[Y]$ tels que la valeur de $\ell([Y])$ n'est pas correcte.

- Ou bien il existe un non terminal $[Y]$ tel que la valeur $\ell([Y])$ n'est pas correcte (i.e., elle peut être encore mise à jour), mais pour toutes les règles de production de la forme $[Y] \to \gamma_1 | \ldots | \gamma_m$ (chaque $[Y] \to \gamma_i$ est une règle de production) tous les non terminaux $[Y']$ dans $\gamma_1, \ldots, \gamma_m$ ont déjà les valeurs $\ell([Y'])$ correctes. Donc, $\ell([Y])$ sera mise à jour à la prochaine itération.
- Ou bien, pour chaque non terminal $[Y]$ avec une valeur $\ell([Y])$ incorrecte, il y a au moins un non terminal $[Y']$ dans chacune de ses règles de production $[Y] \to \gamma_1 | \ldots | \gamma_m$ qui a une valeur $\ell([Y'])$ incorrecte.
  - Ou bien chaque γ_i contient un non terminal qui ne génère aucun mot, et donc $[Y]$ ne génère aucun mot, ce qui implique que la valeur $\ell([Y]) = \infty$ est correcte (contradiction).
  - Ou bien tous les non terminaux dans chaque γ_i génèrent un mot. Cependant, dans l'arbre de dérivation du mot le plus court généré par un non terminal, aucun non terminal n'apparaît deux fois ou plus dans la même branche, car autrement un mot plus court pourrait être généré en n'utilisant ce non terminal qu'une seule fois dans la branche concernée. Donc, parmi tous les non terminaux $[Y']$ qui ont une valeur $\ell([Y'])$ incorrecte, il y en a un (notons le $[Y'']$ qui n'utilise pas les autres pour générer le mot le plus court. Soit la valeur $\ell([Y'])$ est déjà correcte (contradiction), ou bien elle sera mise à jour lors de l'itération suivante $(n + 1)$.

□

Complexité de l'algorithme 6. Vu que lors de l'exécution de l'algorithme 6, il y a au plus $|\mathcal{N}|$ itérations de la boucle *Tant que*. La complexité de l'algorithme 6 est bornée par $O(|\mathcal{N}| \times |\mathcal{P}|)$ dans le pire des cas, ce qui correspond à $O(|\mathcal{A}|^8 \times |\mathcal{V}|^4 \times |\mathcal{E}|)$ (resp. $O(|\mathcal{A}|^8 \times |\mathcal{V}|^7)$) dans le modèle du réseau multicouche. Néanmoins, dans une implémentation différente, en distinguant à chaque itération le non terminal $[X]$ qui a la plus petite valeur $\ell([X])$ et en ne mettant plus à jour sa valeur (étape (2.1.1) de l'algorithme 6) car elle est correcte, et en maintenant les autres non terminaux (dont les valeurs ne sont pas encore correctes) dans une une file de priorité, on peut réduire la complexité à $O(|\mathcal{P}| \times \log |\mathcal{N}| + t)$, où t est la longueur totale des règles de production [66]. Dans notre grammaire, $t = O(|\mathcal{P}|)$, la complexité de l'algorithme 6 sera donc de $O(|\mathcal{P}| \times \log |\mathcal{N}|)$ suivant cette implémentation. Fredman et Tarjan proposent une implémentation de l'algorithme de Knuth en $O(|\mathcal{N}| \times \log |\mathcal{N}| + t)$ grâce à leurs tas de Fibonacci [45]. Dans notre cas, cela ramène la complexité de l'algorithme 6 à $O(|\mathcal{N}| \times \log |\mathcal{N}| + |\mathcal{P}|)$, c'est-à-dire $O(|\mathcal{A}|^3 \times |\mathcal{V}|^2 \times \log(|\mathcal{A}|^3 \times |\mathcal{V}|^2) + |\mathcal{A}|^5 \times |\mathcal{V}|^2 \times |\mathcal{E}|) = O(|\mathcal{A}|^5 \times |\mathcal{V}|^2 \times |\mathcal{E}|)$ car nous considérons que $|\mathcal{E}| \geq |\mathcal{V}| - 1$. Si la grammaire a été obtenue par l'automate transformé (minimisation du nombre d'encapsulations), la complexité en fonction du réseau est de $O(|\mathcal{A}|^5 \times |\mathcal{V}|^5)$.

Génération du mot le plus court. Il y a de nombreux algorithmes qui permettent de générer uniformément un mot d'une certaine longueur d'une grammaire à contexte libre ([56, 43, 75]. Plus récemment, plusieurs algorithmes de génération non-uniforme d'un mot et suivant une certaine distribution ont également été proposés [37, 49]. Par exemple, l'algorithme *boustrophédon* ainsi que l'algorithme *séquentiel* proposés dans [43] génèrent aléatoirement des *objets combinatoires étiquetés* d'une certaine longueur à partir de n'importe quelle structure décomposable (y compris les grammaires à contexte libre). L'algorithme boustrophédon a une complexité en $O(n \log n)$ (où n est la longueur de l'objet à générer), l'algorithme séquentiel a une complexité en $O(n^2)$ Mais a une meilleure complexité moyenne. Les deux algorithmes utilisent un tableau pré-calculé de taille linéaire en n. Le calcul de ce tableau est en $O(n^2)$. Ces algorithmes nécessitent une grammaire non ambiguë en entrée. Or, la grammaire produite par l'algorithme 5 peut être ambiguë (car plusieurs chemins faisables peuvent avoir la même trace). Néanmoins,

39. Une ε-production est une règle de production de la forme $[X] \to \varepsilon$, où $[X]$ est un non terminal.

la non ambiguïté de la grammaire est nécessaire pour garantir l'uniformité de la génération. Notre but est de générer la plus courte trace (séquence de protocoles) qui nous permettra de calculer le plus court chemin (en nombre de liens ou en nombre d'encapsulations). Nous ne prenons donc pas en considération la probabilité de génération des plus courtes traces quand il y en a plusieurs, et donc nous ne nous intéressons pas aux aspects aléatoires de la génération. Pour générer le mot le plus court appartenant au langage $L(G_{\mathcal{R}})$, l'algorithme boustrophédon peut prendre $G_{\mathcal{R}}$ et $\ell([S_G])$ en entrée (rappelons que $\ell([S_G])$ est la longueur du mot le plus court généré par $G_{\mathcal{R}}$). Donc, la génération du mot le plus court w (resp. w') aurait une complexité en $O(|w|^2)$ (resp. $O(|w'|^2)$), où $|w|$ est la longueur (nombre de symboles) de w. Cette complexité inclut le pré-calcul du tableau évoqué ci-dessus. Cependant, cette complexité cache des « constantes » qui dépendent de la taille de la grammaire. Le pré-calcul du tableau nécessite de lire au moins une fois toutes les règles de production de la grammaire.

Il s'avère que le calcul des valeurs $\ell([X])$ pour tous les non terminaux $[X] \in \mathcal{N}$, effectué par l'algorithme 6, peut nous permettre de générer plus efficacement (linéairement en nombre de non terminaux et longueur du mot) le mot le plus court. Ceci est effectué par l'algorithme 7, qui prend en entrée la grammaire $G_{\mathcal{R}}$ et toutes les valeurs $\ell([X])$. Dans la dérivation, pour toutes les règles de production $[X] \rightarrow \gamma_1|\gamma_2|\ldots|\gamma_m$ ayant$[X]$ comme partie gauche, $[X]$ est remplacé par la partie droite γ_i telle que $\gamma_i = \arg\min\{\ell(\gamma_1),\ldots,\ell(\gamma_m)\}$

Algorithme 7 Calcul du mot le plus court généré par $G_{\mathcal{R}}$

Entrée : $G_{\mathcal{R}} = (\mathcal{N},\Sigma,[S_G],\mathcal{P})$ (resp. $(\mathcal{N},\Sigma',[S_G],\mathcal{P}')$) et toutes les valeurs $\ell([X])$ pour tout $[X] \in \mathcal{N}$.
Sortie : w (resp. w'), le mot le plus court généré par $G_{\mathcal{R}}$
/* $[S_G] \rightarrow \gamma_1|\gamma_2|\ldots|\gamma_m$ *est l'ensemble des règles de production ayant* $[S_G]$ *comme partie gauche.* */
(1) $w \leftarrow \arg\min\{\ell(\gamma_1),\ldots,\ell(\gamma_m)\}$ (resp. w')
(2) Tant qu'il y a au moins un non terminal dans w (resp. w')
  (2.1) Pour chaque non terminal $[X]$ dans w (resp. w') faire
    (2.1.1) Remplacer $[X]$ dans w (resp. w') par $\arg\min\{\ell(\gamma'_1),\ldots,\ell(\gamma'_{m'})\}$
  /* $[X] \rightarrow \gamma'_1|\ldots|\gamma'_{m'}$ *est l'ensemble des règles de production ayant* $[X]$ *comme partie gauche.* */

Complexité de l'algorithme 7. Vu qu'il n'y a pas d'ε-production dans $G_{\mathcal{R}}$ (voir l'algorithme 5), toutes les branches dans l'arbre de dérivation se terminent par un symbole appartenant à Σ (resp. Σ'). La longueur de chaque branche est au pire $|\mathcal{N}|$ (un non terminal ne peut apparaitre deux fois ou plus dans la même branche). Le nombre de dérivations et la complexité de l'algorithme 7 sont bornés par $O(|\mathcal{N}| \times |w|)$ (resp. $O(|\mathcal{N}| \times |w'|)$), ce qui correspond à $O(|\mathcal{A}|^3 \times |\mathcal{V}|^2 \times |w|)$ (resp. $O(|\mathcal{A}|^3 \times |\mathcal{V}|^2 \times |w'|)$) dans le modèle de réseau multicouche.

Exemple. L'algorithme 6 nous permet de calculer que $\ell([S_G]) = 3$. L'algorithme 7 génère le mot le plus court en utilisant les règles de production sur la figure 5.8. La dérivation est la suivante :

$$[S_G] \overset{(1)}{\vdash} [S_A z_0 D_A] \overset{(2)}{\vdash} [U_a z_0 D_A] \overset{(3)}{\vdash} a[V_b a D_a][D_a z_0 D_A] \overset{(4)}{\vdash} a\bar{b}_2[D_a z_0 D_A] \overset{(5)}{\vdash} a\bar{b}_2 a$$

Le mot le plus court accepté par l'automate transformé $A'_{\mathcal{R}}$ est donc $a\bar{b}_2 a$, et la trace du chemin le plus court (en nombre d'encapsulations) est $f(a\bar{b}_2 a) = ab\bar{b}a$.

5.4.5 Du mot le plus court au chemin le plus court

Si le but est de minimiser le nombre de liens dans le chemin, l'algorithme 8 prend en entrée w, le mot le plus court accepté par l'automate original $A_{\mathcal{R}}$. Si le but est de minimiser le nombre d'encapsulations dans le chemin, vu que w' est le mot le plus court accepté par l'automate transformé $A'_{\mathcal{R}}$, selon la proposition 4, $f(w')$ est le mot qui minimise le nombre d'empilements (et de dépilements) pour être accepté par l'automate original $A_{\mathcal{R}}$. Dans ce cas, $f(w')$ est bien la trace T_C du chemin faisable C qui minimise le nombre d'encapsulations dans le réseau multicouche $\mathcal{R}$. Il est possible que plusieurs chemins correspondent à cette trace $T_C = w$ (resp. $f(w')$). Dans ce cas, un chemin peut être choisi aléatoirement ou selon une politique d'équilibrage de charge.

L'algorithme 8 est un algorithme de programmation dynamique qui calcule le chemin C. Il démarre du nœud S et sélectionne à chaque étape les liens dans E qui correspondent au protocole (symbole) courant dans T_C. Soit $T_C = x^1x^2 \dots x^n$ ($x^i \in \mathcal{A} \cup \overline{\mathcal{A}}$). A chaque étape i, l'algorithme démarre de chaque nœud U dans $Nodes[i]$ et ajoute à $Links[i]$ tous les liens (U,V) qui correspondent à x^i. Il ajoute également V à $Nodes[i+1]$. Quand il atteint le nœud D, il revient en arrière en sélectionnant les liens de D à S.

Algorithme 8 Calcul du plus court chemin à partir de la plus courte trace

Entrée : Le réseau $\mathcal{R}$ et la trace T_C
Sortie : Le chemin le plus court C
(1) $Nodes[1] \leftarrow S$; $i \leftarrow 1$
(2) Tant que le nœud D n'est pas atteint faire
  (2.1) Pour chaque nœud $U \in Nodes[i]$ et chaque nœud $V \in \mathcal{V}$ tels que (U,V) $\in \mathcal{E}$ faire
    (2.1.1) Si $x^i \in \mathcal{A}$, $x^i \in Out(U)$, $x^i \in In(V)$ et (x^{i-1},x^i) $\in \wp(U)$
      (2.1.1.1) Ajouter (U,V) à $Links[i]$ et V à $Nodes[i+1]$
    (2.1.2) Si $x^i \in \overline{\mathcal{A}}$, $x^i \in Out(U)$, $x^i \in In(V)$ et $\overline{(x^i,x^{i-1})} \in \wp(U)$
      (2.1.2.1) Ajouter (U,V) à $Links[i]$ et V à $Nodes[i+1]$
  (2.2) $i++$
(3) Revenir en arrière de D jusqu'à S en ajoutant chaque lien parcouru à C.

Complexité de l'algorithme 8. La boucle *Tant que* se termine après exactement $|T_C|$ itérations, étant donné qu'il est certain qu'il y a un chemin faisable de longueur $|T_C|$ dans le réseau si T_C est acceptée par l'automate $A_{\mathcal{R}}$. A chaque itération, tous les liens et les nœuds sont parcourus dans le pire des cas. L'algorithme 8 a une complexité en $O(|T_C| \times |\mathcal{V}| \times |\mathcal{E}|)$ dans le pire des cas.

Exemple. A partir de la plus courte trace $ab\bar{b}a$, l'algorithme 8 calcule le seul chemin faisable dans le réseau sur la figure 5.5, qui est S,a,U,b,V,$\bar{b}$,W,a,D.

5.5 Généralisation à n'importe quelle mesure additive

Dans la précédente section, nous avons présenté une approche algorithmique permettant de calculer un chemin minimisant le nombre de liens ou le nombre d'encapsulations dans un réseau multicouche. Dans cette section, nous nous proposons de généraliser cette approche pour calculer le chemin de longueur minimum selon n'importe quelle mesure additive. Nous associons un *poids* (ou *coût*) à chaque triplet (U,β,V) où (U,V) est un lien dans $\mathcal{E}$ et β une fonction d'adaptation dans $\wp(U)$. La somme de ces poids le long d'un chemin sera le poids de ce chemin. Le problème est donc de trouver le chemin dont le poids est le plus petit. L'approche utilisée sera la même, le réseau sera converti en automate à pile, mais nous définirons une fonction de poids sur les transitions. De même, lors de la conversion de l'automate à pile en grammaire à contexte libre, une fonction de poids équivalente sera définie sur les règles de productions de la grammaire. La trace du chemin de poids minimum sera générée, puis le chemin correspondant à cette trace sera calculé.

Il s'avère que la plupart des algorithmes décrits dans la section précédente ne nécessitent que des modifications mineures. Seul les algorithmes 6 et 8 doivent être modifiés en profondeur.

5.5.1 Ajout de la fonction de poids au modèle d'un réseau multicouche

Un réseau multicouche est défini comme étant un quadruplet $\mathcal{R} = (\mathcal{G},\mathcal{A},\wp,h)$ où :
- $\mathcal{G} = (\mathcal{V},\mathcal{E})$ est un graphe orienté représentant la topologie du réseau. Nous distinguons toujours la source S et la destination D,
- $\mathcal{A} = \{a,b,c,\dots\}$ est un alphabet fini dont chaque symbole représente un protocole,
- $\wp$ est l'ensemble des fonctions d'adaptations (tel que défini dans la section 5.3). Les définitions de $In(U)$,$Out(U)$,$\wp(U)$,$\mathcal{T}(U)$ pour un nœud $U \in \mathcal{V}$ restent les mêmes, de même que les définitions de $\mathcal{E}D$ et $\overline{\mathcal{A}}$,

– $h : \mathcal{V} \times \wp \times \mathcal{V} \to \mathbb{R}_+$ est la fonction de poids. La valeur $h(U,\beta,V)$ (où $U,V \in \mathcal{V}$ et $\beta \in \wp$) est le coût d'application de la fonction d'adaptation β dans le nœud U puis la transmission des PDU au nœud V. La fonction h permet ainsi d'associer un coût à chaque lien (indifféremment de la fonction d'adaptation utilisée) ou bien définir un coût sur les liens et les fonctions d'adaptation (c'est-à-dire que pour un même lien $(U,V) \in \mathcal{E}$, la valeur de $h(U,\beta,V)$ ne sera pas forcément la même selon β).

La définition de la faisabilité d'un chemin reste la même que précédemment. Nous étendons le domaine de la fonction h à l'ensemble des chemins faisables. Soit C un chemin faisable tel que $C = Sx^1U_1x^2 \dots x^iU_ix^{i+1} \dots U_{n-1}x^nD$, nous définissons $h(C)$ comme étant :

$$h(C) = \sum_{i=0}^{n-1} h(U_i,\beta_i,U_{i+1}) \tag{5.1}$$

où $U_0 = S$, $U_n = D$ et $\beta_i \in \wp(U_i)$ est la fonction d'adaptation utilisé par U_i pour adapter le protocole x^i en protocole x^{i+1}. La fonction d'adaptation peut être :

– $\beta_i = (x^i,x^{i+1})$ (encapsulation) si $x^i \in \mathcal{A}$ et $x^i \neq x^{i+1}$ et $x^i \neq \overline{x}^{i+1}$,
– $\beta_i = (x^i,x^i)$ (transition passive) si $x^i = x^{i+1}$ ou $x^i = \overline{x}^{i+1}$,
– $\beta_i = (x^i,x^{i+1})$ (désencapsulation) si $x^i \in \overline{\mathcal{A}}$.

β_0 est une fonction d'adaptation équivalente à (x^1,x^1) que nous ajoutons par convenance.

Le problème est donc le suivant :

$$\begin{array}{l} \min\ h(C) \\ t.q. \left\{ \begin{array}{l} C \text{ est un chemin faisable de } S \text{ à } D \end{array} \right. \end{array} \tag{5.2}$$

5.5.2 Conversion du réseau en automate à pile pondéré

Nous définissons un automate à pile pondéré de la même manière qu'en section 5.4.2, excepté que nous ajoutons une fonction de poids sur les transitions. Plus formellement, un automate à pile pondéré est un octuplet $A^w = (\mathcal{Q},\Sigma,\Gamma,\delta,S_A^w,z_0,\mathcal{F},\omega)$ où :

– $\mathcal{Q}$ est l'ensemble des états,
– Σ est l'alphabet en lecture,
– Γ est l'alphabet de la pile,
– δ est l'ensemble des transitions,
– S_A^w est l'état initial,
– z_0 est le symbole de fond de pile,
– $\mathcal{F} \subset \mathcal{Q}$ est l'ensemble des états finaux,
– $\omega : \delta \to \mathbb{R}_+$ est une fonction de poids sur les transitions de l'automate.

L'automate pondéré correspondant au réseau $\mathcal{R}$, noté $A_{\mathcal{R}}^w = (\mathcal{Q},\Sigma,\Gamma,\delta,S_A^w,z_0,\{D_A^w\},\omega)$, est construit en appliquant l'algorithme 3, en y ajoutant les instructions suivantes :

1. Pour chaque transition de la forme $t = (S_A^w,\langle\epsilon,z_0,z_0\rangle,U_x)$ avec $x \in Out(S)$, faire $\omega(t) \leftarrow 0$,
2. Pour chaque transition de la forme $t = (D_x,\langle x,z_0,\emptyset\rangle,D_A^w)$ avec $x \in InD)$, faire $\omega(t) \leftarrow 0$,
3. Pour chaque transition $t = (U_x,\langle x,\alpha,\alpha'\rangle,V_y)$ (avec $U_x,V_y \in \mathcal{Q} \setminus \{S_A^w,D_A^w\}$) faire :
   – Si $\alpha = \alpha'$ (transition passive) alors $\omega(t) \leftarrow h(U,(\alpha,\alpha),V)$,
   – Si $\alpha = x\alpha'$ (empilement de x) alors $\omega(t) \leftarrow h(U,(\alpha,x),V)$,
   – Si $\alpha' = \emptyset$ (dépilement de α) alors $\omega(t) \leftarrow h(U,\overline{(\alpha,x)},V)$,

En plus des instructions de l'algorithme 3, nous affectons à chaque transition créée le poids du lien et de la fonction d'adaptation qui en sont à l'origine. Nous affectons aux transitions fictives ajoutées par convenance un poids nul (étapes 1 et 2 ci-dessus). Pour toute autre transition qui modélise un lien (U,V) et une fonction d'adaptation β, nous affectons le poids $h(U,\beta,V)$. Ces nouvelles instructions ne modifient pas l'état des variables utilisées dans l'algorithme 3, sa version modifiée reste donc correcte. De plus, il n'y a au plus qu'un seul triplet *(nœud, fonction d'adaptation, nœud)* à l'origine de chaque transition, les poids des transitions sont donc bien définis.

Les instructions supplémentaires ci-dessus peuvent être effectuées à la création de chaque transition. La complexité de l'algorithme reste donc asymptotiquement la même.

Remarque. Dans la littérature, les automates à pile pondérés sont définis comme un sextuplet $A^w = (\mathcal{Q},\Gamma,M,S_A^w,z_0,\mathcal{F})$ où M, appelée *matrice de transition empilement/dépilement*[40], est une matrice définie sur un semi-anneau de séries formelles. Ceci permet d'exprimer l'alphabet Σ, l'ensemble de transitions δ et la fonction de poids ω comme étant une seule entité $M \in ((R\langle\langle\Sigma^*\rangle\rangle)^{\mathcal{Q}\times\mathcal{Q}})^{\Gamma^*\times\Gamma^*}$ où $R\langle\langle\Sigma^*\rangle\rangle$ est l'ensemble de toutes les séries formelles de Σ sur un semi-anneau R. Par souci de simplification et pour pouvoir réutiliser les définitions de la section 5.4, nous avons défini un automate à pile pondéré comme un automate à pile classique augmenté d'une fonction de poids sur l'ensemble des transitions. Concernant les fondements théoriques des automates à pile pondérés, le lecteur intéressé peut consulter l'ouvrage de référence de Petre et Salomaa [91].

5.5.3 Conversion de l'automate à pile pondéré en grammaire pondérée

Suivant la même méthode que dans la section 5.4, nous allons convertir l'automate à pile pondéré en grammaire à contexte libre pondérée. Une grammaire à contexte libre pondérée est définie comme une grammaire à contexte libre classique augmentée d'une fonction de poids sur les règles de production. Formellement, nous allons convertir l'automate $A_{\mathcal{R}}^w$ end grammaire $G_{\mathcal{R}}^w = (\mathcal{N},\Sigma,[S_{G^w}],\mathcal{P},\pi)$ où l'ensemble des non terminaux $\mathcal{N}$, l'alphabet Σ, l'axiome $[S_{G^w}]$ et l'ensemble des règles de production $\mathcal{P}$ sont définis de manière analogue à la section 5.4.4.1. La fonction $\pi : \mathcal{P} \to \mathbb{R}_+$ associe un poids à chaque règle de production.

La grammaire $G_{\mathcal{R}}^w$ sera calculée avec l'algorithme 5, avec quelques instructions supplémentaires. Nous rappelons que dans l'algorithme 5, chaque transition de l'automate est à l'origine de la création d'une ou de plusieurs règles de production. Mais une règle de production est issue d'une seule transition de l'automate. Pour que le poids des règles de production soit bien défini, il suffit donc d'associer à chaque règle de production le poids de la transition qui en est à l'origine, i.e., si $r \in \mathcal{P}$ modélise la transition t, alors $\pi(r) \leftarrow \omega(t)$. Ces affectations peuvent se faire au fur et à mesure de la création des règles de production, la complexité de l'algorithme reste asymptotiquement identique.

5.5.4 Génération de la trace de poids minimum

5.5.4.1 Calcul des valeurs de poids associées à chaque non terminal

Les poids des règles de production n'étant plus unitaires, il faut redéfinir la fonction ℓ présentée en section 5.4.4.2. Soit $[X] \in \mathcal{N}$ un non terminal. Nous redéfinissons la fonction $\ell([X])$ comme étant la plus petite somme des poids des règles de production qui, partant du non terminal $[X]$, permettent de dériver (générer) un mot dans Σ^*. La valeur $\ell([S_{G^w}])$ sera donc le poids correspondant à la dérivation de la trace T_C, où C est le chemin de poids minimum.
La fonction $\ell(.)$ est redéfinie de la manière suivante :
$\ell : \{\mathcal{N} \cup \Sigma \cup \{\epsilon\}\}^* \to \mathbb{R}_+ \cup \{\infty\}$ telle que :

- Si $w = \epsilon$ ou $w \in \Sigma$ alors $\ell(w) = 0$,
- Si $w = \alpha_1 \dots \alpha_n$ (avec $\alpha_i \in \{\mathcal{N} \cup \Sigma \cup \{\epsilon\}\}$) alors $\ell(w) = \sum_{i=1}^{n} \ell(\alpha_i)$.
- Soit $r_1 = [X] \to \gamma_1, r_2 = [X] \to \gamma_2, \dots, r_k = [X] \to \gamma_k$ (avec $\gamma_i \in \{\mathcal{N} \cup \Sigma \cup \{\epsilon\}\}^*$) l'ensemble des règles de production ayant le non terminal $[X]$ comme partie gauche. Alors $\ell([X]) = \min\{\pi(r_1) + \ell(\gamma_1), \dots, \pi(r_k) + \ell(\gamma_k)\}$

L'algorithme 9 est une généralisation de l'algorithme 6 mais reste un cas particulier de l'algorithme général de Knuth [66]. Le principe de l'algorithme reste le même sauf que des poids possiblement différents sont attribués aux règles de production, et que ce poids ne dépend plus du nombre de terminaux générés par une règle de production.

La complexité de l'algorithme reste la même que celle de l'algorithme 6 car la modification ne concerne que la formule de calcul des valeurs de $\ell([X])$.

5.5.4.2 Génération de la trace de poids minimum.

Récemment, plusieurs auteurs ont proposé des algorithmes de génération de mots à partir d'une grammaire pondérée [37, 49]. Néanmoins, leur définition d'une grammaire pondérée n'est pas la même que la nôtre, la fonction de poids étant définie sur les non terminaux et non sur les règles de production. Nous ne pouvons donc pas utiliser ces algorithmes.

40. Push-Down transition matrix

Algorithme 9 Calcule la valeur $\ell([X])$ pour chaque non terminal $[X] \in \mathcal{N}$

Entrée : $G_N = (\mathcal{N},\Sigma,[S_{G^w}],\mathcal{P},\pi)$
Sortie : $\ell([X])$ pour chaque non terminal $[X]$
(1) Initialiser toutes les valeurs $\ell([X])$ à ∞
(2) Tant qu'une valeur $\ell([X])$ a encore été mise à jour à la précédente itération
  (2.1) Pour chaque règle de production $r = [X] \rightarrow \alpha_1 \dots \alpha_n$ dans $\mathcal{P}$
    (2.1.1) $\ell([X]) \leftarrow \min\{\ell([X]),\pi(r) + \sum_{i=1}^{n} \ell(\alpha_i)\}$

Pour générer la trace de poids minimum, il suffit d'appliquer l'algorithme 7 qui effectuera la dérivation de poids minimum. La seule différente par rapport au cas non pondéré est que les valeurs de $\ell[X]$ ne vont pas être les mêmes, car elles seront calculées différemment (par l'algorithme 9).

5.5.5 Calcul du chemin de poids minimum à partir de sa trace

Une fois la trace T_C du chemin de poids minimum C obtenue, il faut retrouver le chemin (ou un des chemins, s'il y en a plusieurs) qui correspond à cette trace. L'algorithme 8 peut-être utilisé, il faudra cependant garder les poids en mémoire et les attribuer aux liens. Pour cela, nous ajoutons la structure de données $weight[]$ qui pour chaque lien (U,V) et chaque valeur de i gardera en mémoire le poids du lien (U,V) et de la fonction d'adaptation appliquée à l'étape i, i.e., la valeur $weight[i,(U,V)] = h(U,(x^{i-1},x^i),V)$ ou $h(U,\overline{(x^i,x^{i-1})},V)$ selon le cas.

L'étape (3) de l'algorithme devra être remplacée par un algorithme de calcul de chemins classique. En effet, il faudra calculer le chemin de poids minimum entre $Nodes[1]$ (qui est le nœud source) et $Nodes[n]$ (qui est le nœud destination) via les liens dans $Links[i]$ (un seul lien par valeur de i) de façon à minimiser la somme des poids de ces liens. Le poids d'un lien (U,V) dans $Links[i]$ est gardé dans $Weight[i,(U,V)]$.

Algorithme 10 Calcul du chemin de poids minimum à partir de sa trace

Entrée : Le réseau $\mathcal{R}$ et la trace T_C
Sortie : Le chemin le plus court C
(1) $Nodes[1] \leftarrow S$; $i \leftarrow 1$
(2) Tant que le nœud D n'est pas atteint faire
  (2.1) Pour chaque nœud $U \in Nodes[i]$ et chaque nœud $V \in \mathcal{V}$ tels que $(U,V) \in \mathcal{E}$ faire
    (2.1.1) Si $x^i \in \mathcal{A}$, $x^i \in Out(U)$, $x^i \in In(V)$ et $(x^{i-1},x^i) \in \wp(U)$
      (2.1.1.1) Ajouter (U,V) à $Links[i]$ et V à $Nodes[i+1]$
      (2.1.1.2) $Weight[i,(U,V)] \leftarrow h(U,(x^{i-1},x^i),V)$
    (2.1.2) Si $x^i \in \overline{\mathcal{A}}$, $x^i \in Out(U)$, $x^i \in In(V)$ et $\overline{(x^i,x^{i-1})} \in \wp(U)$
      (2.1.2.1) Ajouter (U,V) à $Links[i]$ et V à $Nodes[i+1]$
      (2.1.2.2) $Weight[i,(U,V)] \leftarrow h(U,\overline{(x^i,x^{i-1})},V)$
  (2.2) $i++$
(3) Calculer le chemin de poids minimum entre $Nodes[1]$ et $Nodes[n]$, l'ensemble des liens étant $Links[i]$ pour toutes les valeurs de i $(1 \leq i \leq n)$.

La complexité des étapes (1) et (2) est en $O(|\mathcal{V}|\times|\mathcal{E}|\times|T_C|)$, comme celle de l'algorithme 8. Néanmoins, l'étape (3) est différente. La structure formée par $Links[]$, $Nodes[]$ et $Weight[]$ est un DAG [41], il suffit donc d'un parcours en largeur pour calculer le chemin le plus court. La complexité du parcours en largeur est linéaire en fonction du nombre de nœuds et de liens. Le nombre de liens est au pire des cas en $O(|\mathcal{E}| \times |T_C|)$ et le nombre de nœuds est au pire en $O(|\mathcal{V}| \times |T_C|)$. La complexité de l'étape (3) est donc en $O((|\mathcal{V}| + |\mathcal{E}|) \times |T_C|)$. La complexité de l'algorithme 10 est donc en $O(|\mathcal{V}| \times |\mathcal{E}| \times |T_C|)$ dans le pire des cas.

41. DAG : Directed Acyclic Graph (Graphe Orienté Acyclique).

5.6 Chemin faisable sous contrainte de bande passante

5.6.1 Définition du problème

L'ajout d'une contrainte de bande passante au calcul de chemins dans un réseau multicouche modifie fondamentalement le problème. En effet, dans un calcul de chemins classique (sans contraintes de compatibilité), il suffit de supprimer les liens dont la bande passante disponible est insuffisante puis de calculer le chemin le plus court dans le réseau restant. L'opération de suppression des liens est appelée *élagage* du réseau. Par extension, on dit que la contrainte de bande passante est *élagable*.

Cette propriété n'est plus vraie sous contraintes de compatibilité. En effet, le chemin faisable le plus court peut comporter des boucles (voir la section 5.2.1.1), et donc des liens par lesquels le chemin passe plusieurs fois. Or, avant de calculer un chemin, on ne sait pas s'il comportera des boucles ni combien de fois il passera par certains liens. On peut toujours élaguer les liens dont la bande passante est insuffisante pour un seul passage. Mais s'il y a plusieurs passages sur un même lien, rien de garantit que la bande passante de ce lien soit suffisante pour tous les passages.

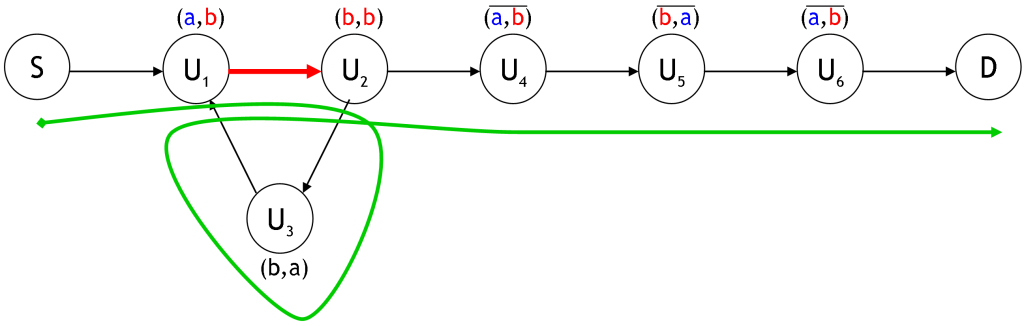


FIGURE 5.9 – Un réseau où le seul chemin faisable entre S et D comporte une boucle (en vert). Le lien (U_1,U_2) (en rouge) est parcouru deux fois par le chemin.

Exemple. La figure 5.9 représente un réseau multicouche avec des fonctions d'encapsulations. Les fonctions d'adaptation (encapsulations, désencapsulation et transition passive) d'un nœud sont notées à côté de celui-ci. Le seul chemin faisable est tracé en vert. Ici, le chemin doit boucler pour accumuler les encapsulations (a,b), (b,a) puis (a,b), et ceci afin de pouvoir traverser les nœuds U_4, U_5 et U_6 qui n'ont que des fonctions de désencapsulation. Imaginons que l'on cherche un chemin faisable tel que sa bande passante soit supérieure à 10Mbps, et imaginons que la bande passante disponible sur le lien (U_1,U_2) (ici en rouge) soit de 15Mbps. Ce lien ne sera pas élagué en amont du calcul de chemin car sa bande passante est suffisante pour un seul passage. Cependant, le chemin faisable doit passer deux fois sur ce lien, et nécessite donc 20Mbps de bande passante disponible. Le chemin viole donc la contrainte de bande passante.

5.6.1.1 Notations et formalisation du problème.

Le modèle de réseau reste le même que précédemment, on y ajoute une mesure de bande passante sur chaque lien. A chaque lien $e = (U,V) \in \mathcal{E}$, on associe sa capacité de bande passante $l_1(e)$. Le nombre de fois qu'un même chemin C passe par un lien e est noté $nb_C(e)$, ou simplement $nb(e)$ s'il n'y a pas d'ambiguïté. La contrainte de bande passante s'exprime par une valeur de bande passante minimum sous laquelle le chemin ne doit pas descendre. Notons cette limite l_1^{min}. Le problème de calculer le plus court chemin faisable sous contraintes de bande passante peut être formalisé ainsi :

$$\begin{aligned} &\min\ h(C) \\ &s.t. \begin{cases} C \text{ est un chemin faisable de } S \text{ à } D \\ \\ \min_{e \in C} \dfrac{l_1(e)}{nb(e)} \geq l_1^{min} \end{cases} \end{aligned} \tag{5.3}$$

5.6.2 Complexité du problème avec deux protocoles

Nous avons montré que le calcul du plus court chemin dans un réseau multicouche sans contrainte de bande passante peut être effectué en temps polynomial. D'un autre côté, le calcul du plus court chemin sous contrainte de bande passante et sans contraintes de compatibilité est trivialement polynomial (il suffit d'élaguer le réseau puis d'effectuer un calcul de chemin classique). Il se trouve que la combinaison des deux problèmes, à savoir l'association d'une contrainte de bande passante et des contraintes de compatibilité, rend le problème NP-complet.

Kuipers et Dijkstra [67] ont montré que le problème de savoir s'il existe un chemin faisable sous contrainte de bande passante dans un réseau multicouche est NP-complet. Ils proposent une réduction polynomiale du problème 1-*in*-3 SAT[42] vers ce problème. Nous améliorons ce résultat en prouvant que le problème est NP-complet avec deux protocoles seulement.

Proposition 6. *Le problème de décision associé au problème 5.3 est* NP*-complet avec deux protocoles.*

Démonstration. Le problème est trivialement dans NP. Ayant un chemin (décrit par une liste de nœuds et de protocoles), on peut vérifier en temps polynomial (et même linéaire), si ce chemin est faisable.

Le reste de la preuve est une réduction polynomiale du problème du chemin hamiltonien [43] vers le problème de chemin faisable sous contrainte de bande passante. Étant donné un graphe orienté $\mathcal{G}' = (\mathcal{V}',\mathcal{E}')$ et un couple de nœuds (S',D'), décider s'il existe un chemin hamiltonien de S' vers D' dans $\mathcal{G}'$ est NP-complet [50].

Une instance du problème du chemin hamiltonien peut être transformée en instance du problème 5.3 en temps polynomial. Le réseau multicouche $\mathcal{R} = (\mathcal{G} = (\mathcal{V},\mathcal{E}),\mathcal{A},\wp)$ correspondant au graphe orienté $\mathcal{G}' = (\mathcal{V}',\mathcal{E}')$ est obtenu de la manière suivante :

- $\mathcal{G} = (\mathcal{V},\mathcal{E})$ est composé d'un chemin $C = S,U_1,\ldots,U_{|\mathcal{V}'|}$, où S est le nœud source et il y a un arc de $U_{|V'|}$ à S'. Chaque nœud $V \in \mathcal{V}'$ est remplacé par deux nœuds V_1 et V_2, tels qu'il y a un arc de V_1 vers V_2, noté $a_+ = (V_1,V_2)$. Tous les arcs entrant dans V' seront transformés en arcs entrants dans V_1 et tous les arcs sortants de V' sont transformés en arcs sortants de V_2. Les figures 5.10 et 5.11 montrent ces transformations. Les fonctions d'adaptation de chaque nœud sont indiquées à côté de celui-ci,
- L'ensemble des protocoles est un alphabet binaire $\mathcal{A} = \{a,b\}$,
- Pour chaque nœud U_i dans le chemin C :
  - Si i est impair, alors U_i peut uniquement encapsuler le protocole a dans le protocole b, i.e., $\wp(U_i) = \{(a,b)\}$,
  - Si i est pair, alors U_i peut uniquement encapsuler le protocole b dans le protocole a, i.e.,$\wp(U_i) = \{(b,a)\}$,

  Pour chaque couple $(V_1,V_2) \in \mathcal{V}^2$, le nœud V_1 peut transmettre passivement n'importe quel protocole (a ou b) et le nœud V_2 peut désencapsuler a depuis b ou bien b depuis a, i.e., $\wp(V_1) = \{(a,a),(b,b)\}$ et $\wp(V_2) = \{(a,a),(b,b)\}$,
- Pour chaque lien (U,V) dans $\mathcal{E}$, $h(U,V) = 1$.
- Pour chaque arc $(U,V) \in \mathcal{E}$, nous posons la contrainte de bande passante $l_1(U,V) = 1$.

Le problème est de décider s'il existe un chemin faisable sous contrainte de bande passante $l_1^{min} = 1$ du nœud S vers le nœud D de longueur arbitraire (il suffit de choisir une longueur assez grande pour permettre le passage par tous les nœuds). La transformation ci-dessus est réalisée en temps polynomial. Il y a $|\mathcal{V}'|$ encapsulations entre S et S_1 (en parcourant le chemin $C = S,U_1,\ldots,U_{|\mathcal{V}'|},S_1$). Le reste du chemin faisable doit donc comporter $|\mathcal{V}'|$ désencapsulations, ce qui veut dire qu'il lui faudra passer par les nœuds $V_2 \in \mathcal{V}$ car ce sont les seuls nœuds qui possèdent des fonctions de désencapsulation. Or, il n'y a qu'un seul arc entrant dans chaque nœud V_2 (l'arc $a_+ = (V_1,V_2)$) et cet arc ne peut être parcouru qu'une seule fois car sa capacité de bande passante $l_1(V_1,V_2) = 1$ est égale à la bande passante maximum autorisée $l_1^{\min} = 1$. Un chemin faisable doit donc passer une et une seule fois par tous les nœuds $V_2 \in \mathcal{V}$, et par conséquent par tous les nœuds V_1 et par tous les arcs $a_+ = (V_1,V_2)$. Or, si nous contractons tous les arcs a_+ (et que nous fusionnons à nouveau tous les nœuds V_1 et V_2) nous obtiendrons un chemin qui passe une et une seule fois par tous les nœuds de $\mathcal{V}'$, ce qui en fait un chemin hamiltonien entre S' et D' dans $\mathcal{G}'$.

42. 1-*in*-3 SAT est une version de $3SAT$ où la proposition est satisfaite si et seulement si un seul littéral de chaque clause est valué à *vrai*.

43. Un chemin hamiltonien est un chemin visitant une et une seule fois tous les nœuds du graphe.

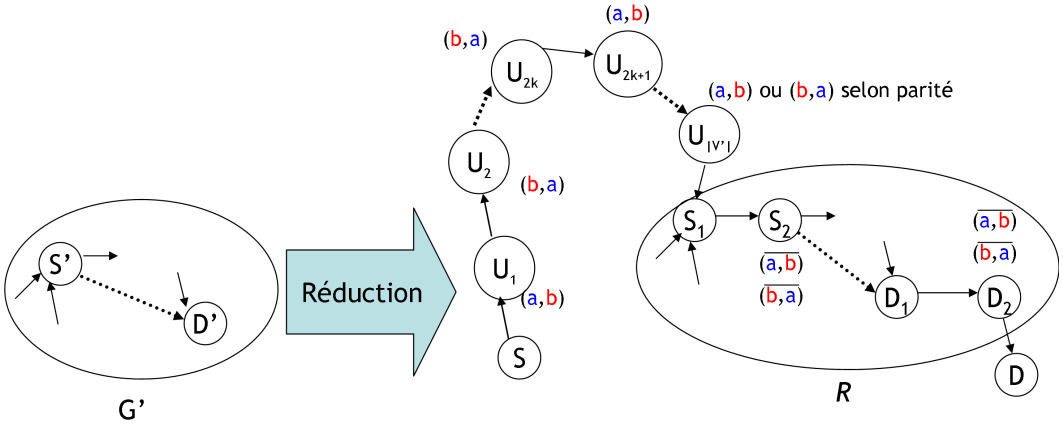


FIGURE 5.10 – Réduction de $\mathcal{G}'$ vers $\mathcal{R}$.

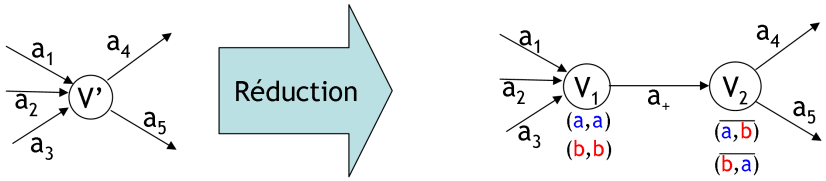


FIGURE 5.11 – Transformation des nœuds et des arcs de $\mathcal{G}'$.

Inversement, supposons qu'il existe un chemin hamiltonien dans $\mathcal{G}'$ entre S' et D'. Notons ce chemin $C' = S', V'^1, \ldots, V'^n, D'$. En transformant chaque nœud V'^i end deux nœuds V_1^i et V_2^i reliés par un arc a_+, nous obtiendrons un chemin passant une et une seule fois par tous les nœuds V_2^i et qui ne passe jamais plus d'une fois par le même arc. Ce chemin, notons-le C'', comprend $|\mathcal{V}'|$ encapsulations. En mettant bout à bout le chemin C et le chemin C'', nous aurons un chemin faisable entre S et D car à toutes les encapsulations dans C correspondront les désencapsulations dans C'', et aucun arc ne sera parcouru plus d'une fois. Ce qui clôt la démonstration de la proposition 6. □

5.6.3 Algorithme de résolution

L'algorithme de Kuipers et Dijkstra [67] permet de calculer un chemin faisable sous contrainte de bande passante. L'algorithme est une exploration en largeur de tous les chemins possibles jusqu'à en trouver un qui soit faisable. Sa complexité dans le pire des cas est exponentielle. Nous ne présentons pas cet algorithme car il se déduit très facilement de l'algorithme plus général que nous présentons en section 5.7. Il suffira de ne retenir que les contraintes de compatibilité et de bande passante et d'ignorer les autres contraintes de QoS.

5.7 Calcul de chemins avec contraintes de Qualité de Service

Dans cette section, nous traitons du problème de calculer un chemin faisable (donc sous contraintes de compatibilité), sous contraintes de bande passante ainsi que plusieurs contraintes additives de QoS. Nous ajouterons donc les différentes mesures de QoS sur les liens. La complexité du problème sera discuté et nous proposerons une adaptation de l'algorithme SAMCRA au contexte multicouche.

5.7.1 Modèle et formalisation du problème

Soir $\mathcal{R}$ un réseau multicouche. A chaque lien $e = (U,V)$ est associé un ensemble de m mesures de QoS $l(e) = (l_1(e), \ldots, l_m(e))$. La première mesure $l_1(e)$ est la bande passante disponible sur le lien, comme

vu dans la précédente section. Les autres mesures $l_i(e)$ for $i = 2 \dots m$ sont des mesures additives qui peuvent correspondre au délai, au logarithme du taux de perte, etc.

Comme il est toujours possible que le seul chemin faisable dans un réseau multicouche comporte des boucles, la contrainte de bande passante n'est pas non plus élagable dans ce cas.

Soit $l^{max} = (l_1^{min} l_2^{max} \dots, l_m^{max})$ un vecteur de contraintes de QoS. Le problème de calculer le plus court chemin faisable sous contraintes de QoS peut être formalisé ainsi :

$$\min h(C) \quad s.t. \begin{cases} C \text{ est un chemin faisable entre } S \text{ et } D \\ \min_{e \in C} \dfrac{l_1(e)}{nb(e)} \geq l_1^{min} \\ \sum_{e \in C} (l_i(e) \times nb(e)) \leq l_i^{max},\ i = 2 \dots m \end{cases} \tag{5.4}$$

Il est bien connu que la version décisionnelle du problème de calcul de chemins sous plusieurs contraintes de QoS est NP-complet, même avec deux contraintes additives et/ou multiplicatives [113] (il est trivial de convertir des contraintes multiplicatives en contraintes additives via une fonction logarithme). Van Mieghem *et al.* [83] ont proposé un algorithme en temps exponentiel pour résoudre le problème. Ils ont montré que les instances nécessitant réellement un temps d'exécution exponentiel sont peu fréquentes et que dans la majorité des cas, l'algorithme s'exécute en temps polynomial.

Le calcul de chemins sous contraintes de QoS est un cas particulier du Problème 5.4 où il n'y a qu'un seul protocole dans tout le réseau et pas d'encapsulations ni désencapsulations. De fait, le problème de décision associé au Problème 5.4 est donc également NP-complet.

5.7.2 L'algorithme SAMCRA

Etant donné que le problème de décision associé au calcul de chemins dans un réseau multicouche avec QoS est NP-complet, tout algorithme de résolution exacte est exponentiel dans le pire des cas (sous réserve que P $\neq$ NP). Néanmoins, le même problème sans contraintes de compatibilité (avec uniquement des contraintes de QoS), bien qu'également NP-complet [83], admet un algorithme de résolution qui a une complexité moyenne [44] polynomiale. L'algorithme le plus connu est l'algorithme *SAMCRA* proposé par Kuipers et Van Mieghem [82].

SAMCRA est un algorithme exact de calcul de chemins sous plusieurs contraintes de QoS additives. Sa complexité dans le pire des cas est exponentielle, mais sa complexité moyenne est polynomiale (sur une distribution uniforme de graphes aléatoires et de contraintes de QoS dans un intervalle fixe). Nous nous proposons d'adapter SAMCRA pour qu'il prenne en compte les contraintes de compatibilité. Nous allons d'abord présenter SAMCRA et ses principaux concepts, ensuite nous proposerons certaines modifications pour qu'il prenne en compte les encapsulations et désencapsulations.

5.7.2.1 Les principaux concepts de SAMCRA

Le principe de SAMCRA est de maintenir une liste de chemins à partir d'un nœud source S vers tous les autres nœuds jusqu'à ce qu'il atteigne le nœud destination D. Il supprime progressivement de la liste tous les chemins qui ne respectent pas les contraintes de QoS spécifiées. Les principaux concepts de SAMCRA sont :

- *Longueur de chemin non linéaire :* Les créateurs de SAMCRA ont considéré un problème de calcul de chemins avec contraintes de QoS légèrement différent du notre. En effet, dans leur définition, les liens n'ont pas de poids ou coûts comme définis dans notre approche par la fonction $h(.)$. Ils disposent seulement des mesures additives de QoS. Dès lors, calculer le plus court chemin n'est pas trivial car il faut définir la longueur d'un chemin. Ils proposent donc de définir la longueur d'un chemin comme étant une fonction non linéaire des paramètres de QoS de chaque lien dans le chemin. Ce concept permet de réduire l'ensemble des solutions à explorer, néanmoins, l'algorithme

44. La complexité temporelle moyenne d'un algorithme est la moyenne de sa complexité temporelle sur une distribution probabiliste de ses données en entrée. Elle décrit le comportement moyen de l'algorithme.

fonctionne également avec n'importe quelle métrique de longueur de chemin. Pour notre part, nous n'avons pas à redéfinir la longueur d'un chemin car elle est explicitement définie par la fonction $h(.)$.

- *L'algorithme des k plus courts chemins :* Il s'agit de maintenir la liste des chemins qui ne sont pas encore dominés et de les classer par longueur. Dans SAMCRA, le fait que k ne soit pas fixé et qu'il puisse être aussi grand que l'on veut est une condition nécessaire à l'exactitude de l'algorithme. Pour chaque nœud U, on doit maintenir la liste de tous les chemins non-dominés entre la source S et le nœud U. Cette restriction est liée à la définition d'une longueur de chemin non-linéaire. Dans notre cas, il suffira simplement de maintenir une liste de chemins non-dominés de la source S vers n'importe quel autre nœud U, jusqu'à atteindre le nœud destination D. Ce nombre de chemins peut-être exponentiel, c'est cela qui ne permet pas d'avoir un algorithme polynomial.
- *Non-dominance :* Un chemin multicontraint C *domine* un autre chemin C' si $\forall i, \sum_{e \in C} l_i(e) \leq \sum_{e \in C'} l_i(e)$ (i.e., si C est meilleur que C' en ce qui concerne tous les paramètres de QoS). Un chemin C est non-dominé s'il n'y a aucune chemin qui le domine. Le concept de non-dominance induit un ordre partiel sur sur l'ensemble des chemins. Il évite l'exploration de beaucoup de chemins et donc réduit substantiellement la complexité de SAMCRA.

La longueur d'un chemin n'est pas impactée par le contexte multicouche et l'utilisation d'une fonction de longueur linéaire est possible. L'algorithme des k plus court chemins n'est pas impacté non plus par les contraintes de compatibilité. Cependant, la non-dominance doit être redéfinie pour prendre en compte les contraintes de compatibilité et les possibles boucles dans les chemins.

5.7.3 Modification de SAMCRA

Un chemin (pas forcément faisable) dans un réseau multicouche est caractérisé par la liste des nœuds parcourus, mais également par la pile de protocole à chaque nœud. Dans la liste de chemins de SAMCRA, chaque chemin doit être enregistré avec sa pile de protocole à son nœud de destination. Comme dit précédemment, un chemin dans un réseau multicouche peut contenir des boucles et ainsi parcourir plusieurs fois le même lien. Avant de vérifier si le chemin respecte les contraintes de bande passante, les paramètres de QoS de chaque lien dans le chemin doivent être multipliés par le nombre de fois que ce lien est parcouru. Par exemple, si un lien e a un délai de 30ms et que le chemin C parcourt ce lien $nb_C(e) = 3$ fois, le délai induit par e dans C n'est pas de 30ms mais 90ms. Comme dit précédemment, la contrainte de bande passante n'est plus égalable à partir du moment où il peut y avoir des boucles. La nouvelle définition de la non-dominance doit prendre en compte ce fait. Nous définissons la non-dominance dans un contexte multicouche de la manière suivante :

Un chemin C domine un chemin C' si :

- $\min_{e \in C} (l_1(e)/nb_C(e)) \geq \min_{e \in C'} (l_1(e)/nb_{C'}(e))$ (nous rappelons que $l_1(e)$ est la bande passante disponible sur le lien e),
- $\sum_{e \in C}(l_i(e) \times nb_C(e)) \leq \sum_{e \in C'}(l_i(e) \times nb_{C'}(e))$, $\forall i = 2, \ldots, m$
- C et C'ont le même nœud final (dernier nœud du chemin),
- Et C et C' ont la même pile de protocoles à leur nœud final.

Avec cette nouvelle définition de la dominance, SAMCRA explore tous les chemins possibles jusqu'à atteindre le nœud destination avec des paramètres de QoS optimaux. Tout au long de l'exploration, il supprime tous les chemins dominés ou non faisables.

5.8 Conclusion

Dans un contexte inter-domaine, le calcul de chemins dans un réseau ne peut pas se réduire au calcul de chemins dans un graphe si l'on prend en compte les aspects hétérogènes des réseaux et les différentes fonctions d'encapsulation et de désencapsulation. Une solution établissant des connexions via plusieurs domaines doit assurer cette prise en compte, car il est utopique de croire que tous les domaines vont utiliser les mêmes technologies et les mêmes protocoles. Il est donc important de formaliser précisément les contraintes induites par ces fonctions et de disposer d'algorithmes efficaces pour calculer des chemins respectant ces contraintes.

Dans ce chapitre, nous avons présenté les spécificités et caractéristiques des chemins dans les réseaux multicouches. Nous avons vu que ces chemins ne sont pas triviaux du tout : les chemins les plus courts peuvent comporter des boucles et n'ont pas une sous-structure optimale. Dès lors, la programmation

Algorithme	Borne supérieure de complexité	
	Minimisation des liens	Minimisation des encapsulations
Algo. 3 : Réseau multicouche → Automate	$O(\|\mathcal{A}\|^3 \times \|\mathcal{E}\|)$	
Algo. 4 : Transformation de l'automate	/	$O(\max(\|\mathcal{A}\|^4 \times \|\mathcal{E}\|, \|\mathcal{A}\|^3 \times \|\mathcal{V}\|^3))$
Algo. 5 : Automate → Grammaire	$O(\|\mathcal{A}\|^5 \times \|\mathcal{V}\|^2 \times \|\mathcal{E}\|)$	$O(\|\mathcal{A}\|^5 \times \|\mathcal{V}\|^5)$
Algo. 6 : Longueur du mot le plus court[45]	$O(\|\mathcal{A}\|^8 \times \|\mathcal{V}\|^4 \times \|\mathcal{E}\|)$ $O(\|\mathcal{A}\|^5 \times \|\mathcal{V}\|^2 \times \|\mathcal{E}\|)$	$O(\|\mathcal{A}\|^8 \times \|\mathcal{V}\|^7)$ $O(\|\mathcal{A}\|^5 \times \|\mathcal{V}\|^5)$
Algo. 7 : Génération du mot le plus court	$O(\|\mathcal{A}\|^3 \times \|\mathcal{V}\|^2 \times \|w\|)$	$O(\|\mathcal{A}\|^3 \times \|\mathcal{V}\|^2 \times \|w'\|)$
Algo. 8 : Le chemin le plus court	$O(\|T_C\| \times \|\mathcal{V}\| \times \|\mathcal{E}\|)$	

TABLE 5.1 – Les algorithmes proposés et leur complexité.

dynamique, qui est à la base de l'efficacité de la plupart des algorithmes de calcul de chemins, ne peut plus être utilisée aussi efficacement. Nous avons fait un tour d'horizon des travaux et des algorithmes actuels de calcul de chemins dans un contexte multicouche.

Nous avons défini un chemin comme étant *faisable* s'il respecte la continuité protocolaire et les contraintes imposées par les fonctions d'encapsulation et de désencapsulation ainsi que leurs emplacements. Nous avons formalisé et caractérisé la faisabilité d'un chemin avec des outils de la Théorie des Langages. En effet, la suite des fonctions d'encapsulation et de désencapsulation impliquées dans un chemin faisable doit être un langage bien parenthésé.

Bien que notre but fût de résoudre le problème du plus court chemin faisable sous plusieurs contraintes de QoS, il s'est avéré qu'aucune solution satisfaisante n'existait pour le calcul de chemins faisables sans aucune contrainte supplémentaire. Nous nous sommes donc intéressé dans un premier temps au calcul du chemin faisable le plus court, que ce soit en nombre de liens ou en nombre de fonctions d'encapsulation, car les fonctions d'encapsulation et de désencapsulation impactent de nombreux paramètres d'un chemin (principalement le délai), d'où l'intérêt de minimiser le nombre de ces fonctions. Nous avons donc utilisé des outils de la Théorie des Langages pour résoudre ce problème : les protocoles sont modélisés par un alphabet, le réseau multicouche est modélisé comme un automate à pile où les encapsulations sont modélisées par des empilements et les désencapsulations par des dépilements. L'automate peut être transformé afin de court-circuiter les séquences sans fonctions d'encapsulation (dans le cas où l'on veut minimiser ces fonctions). L'automate (ou l'automate transformé) est ensuite converti en grammaire à contexte libre, cette grammaire permettant de générer la plus courte séquence de protocoles (considérée comme un mot) correspondant à un chemin faisable. Via la séquence de protocoles la plus courte, un algorithme de programmation dynamique permet de trouver le chemin y correspondant (ou un des chemins, s'il y en a plusieurs qui correspondent à la même suite de protocoles). Le tableau 5.1 donne la complexité en temps des différents algorithmes proposés. Cette solution est la première solution polynomiale obtenue pour le calcul de chemins dans un réseau multicouche.

Nous avons généralisé cette méthode au cas où l'on associe un poids ou un coût à chaque lien et fonction d'adaptation dans le réseau. Cette modification se traduit par l'ajout d'une fonction de poids sur les transitions de l'automate, puis l'ajout d'une fonction identique sur les règles de production de la grammaire correspondant à l'automate. La plupart des algorithmes utilisés dans la version non pondérée du problème restent valables. Seuls deux algorithmes sont à modifier de manière non triviale. La complexité des différents algorithmes reste principalement la même que dans le cas non pondéré, ainsi la solution est également polynomiale.

Le problème change fondamentalement quand on introduit une contrainte de bande passante. Il est bien connu que ce dernier devient NP-complet. Nous avons amélioré ce résultat en prouvant que le problème est NP-complet même s'il y a uniquement deux protocoles.

Enfin, nous nous sommes intéressé au problème général de calcul du plus court chemin faisable sous plusieurs contraintes de QoS (une contrainte de bande passante et plusieurs contraintes additives). Nous avons présenté rapidement l'algorithme SAMCRA, qui est un algorithme bien connu de calcul de chemins sous contraintes de QoS (mais sans contrainte de compatibilité). Bien que SAMCRA ait une complexité exponentielle dans le pire des cas, il s'avère que sa complexité moyenne est polynomiale. Nous avons donc proposé une modification de l'algorithme pour la prise en compte de la faisabilité du chemin et ainsi pouvoir l'utiliser dans un réseau multicouche.

45. La complexité sur la deuxième ligne correspond au cas où on utilise des tas de Fibonacci (voir la section 5.4.4).

Conclusion générale

Afin de permettre la diffusion et la généralisation d'applications multimédias pour les utilisateurs, les opérateurs devront déployer la QoS au sein de leurs propres domaines, mais aussi permettre l'établissement de chemins avec garanties de QoS traversant plusieurs domaines gérés par des opérateurs différents. Ce dernier point pose problème car il nécessite une coopération rendue difficile par les intérêts divergents des opérateurs, mais également par l'hétérogénéité des technologies utilisées par ces opérateurs. Le projet européen ETICS a fourni un cadre économique et technique pour cette coopération. Les solutions proposées se basent sur la mise bout à bout (ou la composition) de SLA. Chaque SLA traversant un domaine, une chaîne de SLA est créée entre le domaine source et le domaine destination.

Dans le processus de calcul de la chaîne de SLA, l'étape de négociation est cruciale. En effet, face à un client qui demande une route (ou une portion de route) avec QoS, le domaine fournisseur doit sélectionner un offre à faire au client. Cette offre, sous forme de SLA, doit respecter la QoS demandée par le client, mais aussi ne pas hypothéquer trop de ressources afin de pouvoir répondre à des offres futures. En cas de concurrence entre plusieurs domaines fournisseurs, chacun doit s'efforcer de ne pas demander un prix trop élevé afin de ne pas être disqualifié par le client. Dans un premier temps, **nous avons proposé un modèle pour les différents acteurs** et paramètres intervenant dans la négociation de SLA. **Nous avons proposé une méthode exacte basée sur les algèbres** $(\max,+)$ **pour calculer la stratégie optimale** d'un domaine en situation de monopole. **Les mêmes outils sont utilisés pour calculer la stratégie optimale de plusieurs domaines qui collaborent**. Nous avons également étudié une situation où les domaines ne collaborent pas et **avons calculé le Prix de l'Anarchie**.

Les méthodes exactes décrites ci-dessus n'étant pas scalables et nécessitant une connaissance parfaites de tous les paramètres en jeu, **nous avons présenté et adapté des algorithmes d'apprentissage au problème de négociation de SLA**. Nous avons ensuite introduit un **mécanisme de réputation et adapté les algorithmes d'apprentissage à ce nouveau mécanisme**. Les simulations montrent que **les algorithmes d'apprentissage sont très efficaces sans nécessiter de connaître tous les paramètres**.

Dans la seconde partie de la thèse, nous nous sommes intéressés au problème de l'hétérogénéité de protocoles qui se pose au moment de l'instanciation de la chaîne de SLA. Nous avons d'abord fait un tour d'horizon des techniques d'encapsulation de protocoles dans différentes architectures, en insistant sur l'architecture Pseudo-Wire. Puis nous avons présenté quelques problèmes de calcul de chemins où la prise en compte de fonctions d'encapsulation est nécessaire.

Nous avons ensuite présenté le problème de manière formelle et expliqué sa relation avec les langages à contexte libre. Cette relation nous a inspiré un modèle de réseau basé sur les automates à pile. **Nous avons proposé plusieurs algorithmes sur ce modèle qui nous ont permis de calculer le chemin le plus court**, que ce soit en nombre de liens ou en nombre d'encapsulations. **La solution que nous avons proposée est la première solution polynomiale à ce problème. Tous nos algorithmes ont été prouvés et analysés afin de donner des bornes supérieures à leur complexité. Nous avons généralisé cette solution au calcul de chemin le plus court sous n'importe quelle mesure additive, puis prouvé que le problème est NP-complet sous contrainte de bande**

passante, ceci même avec deux protocoles uniquement. Enfin, nous avons proposé une adaptation d'un algorithme classique de calcul de chemin avec QoS, afin qu'il prenne en compte les encapsulations de protocoles.

Les travaux effectués dans le cadre de cette thèse se placent à différentes étapes de l'établissement d'un chemin inter-domaine avec garantie de QoS. Ils sont complémentaires car ils proposent des solutions aux deux problèmes rencontrés par le déploiement de QoS en inter-domaine : l'absence de coopération sur le plan économique et l'hétérogénéité des protocoles et technologies sur le plan technique. Bien que ces travaux aient pour but de résoudre des problèmes dans le domaine des réseaux, ils font intervenir des outils théoriques très variés : algèbres $(\max,+)$, chaînes de Markov, processus de décision markoviens, théorie des jeux, algorithmes d'apprentissage, théorie des graphes, langages formels et automates à pile, théorie de la complexité, etc. Le solutions proposées peuvent ainsi avoir un intérêt au-delà du domaine des réseaux.

Perspectives de Recherche

Le sujet du déploiement de la QoS dans les réseaux inter-domaine étant très vaste, nous nous proposons d'aborder uniquement les perspectives dans la continuité directe des travaux de cette thèse :

- Nous avons vu que les algorithmes d'apprentissage permettaient avec plus ou moins de précision de trouver un compromis entre réputation et quantité de ressources utilisées. Néanmoins, nous ne savons pas à l'heure actuelle calculer de façon exacte la valeur de la réputation correspondant à ce compromis. Il serait intéressant de résoudre le problème de manière analytique, et ainsi de calculer la stratégie optimale et la réputation (ou la dynamique de réputation si celle-ci n'est pas constante) maximisant le gain à long terme. Un telle solution ferait intervenir des chaînes de Markov non homogènes, car les transitions entre états, qui sont valuées par une espérance de gains, dépendront du choix du client qui lui-même dépendra de la réputation des domaines à un instant donné.
- Dans les chapitre 2 et 3, nous avons émis l'hypothèse que les requêtes de SLA étaient émises à intervalle régulier. Une autre perspective intéressante serait de considérer diverses lois d'arrivée (poissonnienne, par exemple) et d'étudier leur impact sur les stratégies optimales des domaines.
- Concernant le calcul de chemins avec prise en compte des encapsulations, la première perspective est d'implémenter et de tester les performance de la version modifiée de SAMCRA présentée en section 5.7. Les tests pourraient se faire sur des topologies multicouches réelles. Néanmoins, pour avoir une bonne idée des performances moyennes de l'algorithme, il faudrait le tester sur des topologies multicouches générées aléatoirement. Il existe une théorie très développée sur les graphes aléatoires (modèle de Erdős–Rényi), qui permet entre autre de connaitre la probabilité d'existence d'un chemin entre deux nœuds, la probabilité d'existence de cycle, etc. Cette théorie ne peut malheureusement pas nous servir à générer des réseaux multicouches aléatoires car, comme nous l'avons vu dans le chapitre 5, les graphes finis ne peuvent pas modéliser les fonctions d'encapsulations. Il existe quelques travaux sur les automates à pile probabilistes ou stochastiques qui pourraient servir de base à la génération de réseaux multicouches ayant certaines propriétés (typiquement, la propriété la plus importante est l'existence d'un chemin faisable, car il ne sert à rien de tester l'algorithme sur des topologies où il n'y a pas de chemin faisable).
- Une autre perspective (ou défi) serait de concevoir des versions distribuées des algorithmes présentés dans le chapitre 5. La possibilité de telles solutions n'est pas évidente à première vue, car nous ne savons pas si la connaissance globale de la pile de protocoles est nécessaire (et doit donc être centralisée quelque part) ou si une connaissance uniquement locale (le protocole courant et le sommet de pile) à chaque nœud serait suffisante pour calculer un chemin faisable. L'intérêt de concevoir ces algorithmes distribués réside dans l'implémentation de ces derniers dans un protocole de routage sans entité centralisée de calcul de chemins.

Publications personnelles et brevets

Journal International

[J1] Mohamed Lamine Lamali, Hélia Pouyllau, and Dominique Barth. Path computation in multi-layer multi-domain networks : A language theoretic approach. *Computer Communications*, 36(5), pages 589-599, 2013.

Conférences Internationales

[C1] Mohamed Lamine Lamali, Hélia Pouyllau, Johanne Cohen, Anne Bouillard and Dominique Barth. Risk-aware SLA negotiation. In *VALUETOOLS'13*. 2013.

[C2] Mohamed Lamine Lamali, Hélia Pouyllau, and Dominique Barth. Path Computation in Multi-Layer Multi-domain Networks. In *IFIP Networking*, pages 421-433, 2012.

[C3] Mohamed Lamine Lamali, Hélia Pouyllau, and Dominique Barth. SLA negotiation : Experimental observations on learning policies. In *VALUETOOLS*, pages 243-252, 2012.

[C4] Mohamed Lamine Lamali and Hélia Pouyllau. Game Theory and Reinforcement Learning for inter-provider SLA negotiation (extended abstract). In *3rd International Conference on Control and Optimization with Industrial Applications (COIA)*, 2011.

Workshops internationaux

[W1] Mohamed Lamine Lamali, Dominique Barth, and Johanne Cohen. Reputation-Aware Learning for SLA Negotiation. In *ETICS Workshop (with IFIP Networking)*, pages 80-88, 2012.

[W2] Mohamed Lamine Lamali, Hélia Pouyllau, and Dominique Barth. End-to-end Quality of Service in Pseudo-Wire networks. In *Proceedings of The ACM CoNEXT Student Workshop, CoNEXT '11 Student*, pages 21 :1-21 :2. ACM, 2011.

Brevets

[B1] X. Misseri, J-L. Rougier, H. Pouyllau and M. L. Lamali. 813201-EP-EPA - Method And System For Advertising Inter-Domain Routes - 16/07/2013 - 13176712.1

[B2] H. Pouyllau and M. L. Lamali. 810805-EP-EPA - An Improved Method For Maintaining Traffic Engineering Database In A Network - 13/09/2012 - 12184296.7

[B3] H. Pouyllau and M. L. Lamali. 810804-EP-EPA - An Improved Method For Computing A Constrained Path In A Network - 20/07/2012 - 12177358.4

Références bibliographiques

[1] Cisco Visual Networking Index : Forecast and Methodology, 2013–2018. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html.

[2] La neutralité d'Internet sort renforcée du Parlement européen. http://www.lemonde.fr/technologies/article/2014/04/03/la-neutralite-d-internet-en-jeu-au-parlement-europeen_4394836_651865.html. Article du 3-4-2014.

[3] Le régulateur américain prêt à enterrer la neutralité du Net. http://www.lemonde.fr/economie/article/2014/04/24/le-regulateur-americain-pret-a-enterrer-la-neutralite-du-net_4406592_3234.html. Article du 24-4-2014.

[4] Megaupload : Cogent porte plainte contre Orange. http://www.lemonde.fr/technologies/article/2011/08/29/megaupload-cogent-porte-plainte-contre-orange_1564736_651865.html. Article du 29-8-2011.

[5] Number of ASes in the Internet. http://www.cidr-report.org/as2.0/. Accédé le 14-4-2014.

[6] Number of ASes in the Internet. http://www.caida.org. Accédé le 14-4-2014.

[7] Regulation of the European Parliament and of the Council laying down measures concerning the European single market for electronic communications and to achieve a Connected Continent - COM(2013) 627. https://ec.europa.eu/digital-agenda/en/news/regulation-european-parliament-and-council-laying-down-measures-concerning-european-single.

[8] Site officiel du projet ETICS. https://www.ict-etics.eu/.

[9] VOD : Netflix s'associe avec Comcast. http://www.lemonde.fr/technologies/article/2014/02/24/vod-netflix-s-associe-avec-comcast_4372054_651865.html. Article du 24-2-2014.

[10] IEEE Std 802.3-2005 (Revision of IEEE Std 802.3-2002 including all approved amendments). Technical report, 2005.

[11] ETICS Delivrable D3.5 - Final business models analysis. 2013.

[12] ETICS Delivrable D4.4 - Final ETICS architecture and functional entities high level design. 2013.

[13] Axel Keller & al. Quality assurance of grid service provisioning by risk aware managing of resource failures. In *CRiSIS*, 2008.

[14] Dominic Battré & al. Assessgrid strategies for provider ranking mechanisms in risk-aware grid systems. In *GECON*, 2008.

[15] Alcatel-Lucent. Evolution of the broadband network gateway. http://www3.alcatel-lucent.com/wps/DocumentStreamerServlet?LMSG_CABINET=Docs_and_Resource_Ctr&LMSG_CONTENT_FILE=Application_Notes/Evolution_of_Broadband_Network_EN_AppNote.pdf, 2012.

[16] P. Almquist. RFC1349 - Type of Service in the Internet Protocol Suite, 1992.

[17] Isabel Amigo, Pablo Belzarena, and Sandrine Vaton. On the problem of revenue sharing in multi-domain federations. In *Networking (2)*, pages 252–264, 2012.

[18] ATKearney. Internet value chain economics. `https://www.atkearney.com/fr/paper/-/asset_publisher/dVxv4Hz2h8bS/content/internet-value-chain-economics/10192`, 2010.

[19] F. Baccelli, G. Cohen, G. J. Olsder, and J. P. Quadrat. *Synchronization and Linearity : an algebra for discrete event systems*. 1992.

[20] F. Baker, J. Polk, and M. Dolly. RFC5865 - A Differentiated Services Code Point (DSCP) for Capacity-Admitted Traffic. RFC 5865 (Informational), 2010.

[21] Christopher L. Barrett, Riko Jacob, and Madhav V. Marathe. Formal-language-constrained path problems. *SIAM J. Comput.*, 30(3) :809–837, 2000.

[22] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC 2475 - An Architecture for Differenctiated Services. 1998.

[23] M. Bocci and S. Bryant. RFC5659 - An Architecture for Multi-Segment Pseudowire Emulation Edge-to-Edge, 2009.

[24] Olivier Bonaventure. Using BGP to distribute flexible QoS information. *IETF draft, draft-bonaventure-bgp-qos-00. txt (February)*, 2001.

[25] M Boucadair, P Morand, P Levis, T Coadic, H Asgari, P Georgatsos, D Griffin, D Spencer, and MP Howarth. QoS-enhanced border gateway protocol. 2005.

[26] S. Bryant and P. Pate. RFC3985 - Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture, 2005.

[27] Tian Bu, Lixin Gao, and Don Towsley. On Characterizing BGP Routing Table Growth. *Comput. Netw.*, 45(1) :45–54, May 2004.

[28] R.R. Bush and F. Mosteller. *Stochastic models for learning*. Wiley publications in statistics. Wiley, 1955.

[29] B. Carpenter and C. Jung. Transmission of IPv6 over IPv4 Domains without Explicit Tunnels. RFC 2529 (Proposed Standard), 1999.

[30] I. Chlamtac, A. Faragó, and T. Zhang. Lightpath (Wavelength) Routing in Large WDM Networks. *IEEE Journal on Selected Areas in Communications*, 14(5) :909–913, 1996.

[31] H. Cho, J. Ryoo, and D. King. Stitching dynamically and statically provisioned segments to construct end-to-end multi-segment pseudowires. `http://www.ietf.org/id/draft-cho-pwe3-mpls-tp-mixed-ms-pw-setup-01.txt`, 2011.

[32] Pierre Coucheney, Patrick Maillé, and Bruno Tuffin. Impact of competition between isps on the net neutrality debate. *IEEE Transactions on Network and Service Management*, 10(4) :425–433, 2013.

[33] M. Crawford. RFC2464 - Transmission of IPv6 Packets over Ethernet Networks, 1998.

[34] B. Davie, A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. RFC3246 - An Expedited Forwarding PHB (Per-Hop Behavior). RFC 3246 (Proposed Standard), 2002.

[35] Chrysanthos Dellarocas. Analyzing the economic efficiency of eBay-like online reputation reporting mechanisms. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*, pages 171–179. ACM, 2001.

[36] Haluk Demirkan, Michael Goul, and Daniel S. Soper. Service level agreement negotiation : A theory-based exploratory study as a starting point for identifying negotiation support system requirements. In *HICSS*, 2005.

[37] A. Denise, Y. Ponty, and M. Termier. Controlled non-uniform random generation of decomposable structures. *Theoretical Computer Science*, 411(40-42) :3527–3552, 2010.

[38] Amogh Dhamdhere and Constantine Dovrolis. The internet is flat : modeling the transition from a transit hierarchy to a peering mesh. In *CoNEXT*, page 21, 2010.

[39] Edsger. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 :269–271, 1959.

[40] F. Dijkstra. *Framework for path finding in multi-layer transport networks.* PhD thesis, Universiteit van Amsterdam, June 2009.

[41] Eyal Even-Dar and Yishay Masour. Learning Rates for Q-Learning. *Journal of Machine Learning Research*, 5, 2003.

[42] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The Locator/ID Separation Protocol (LISP). RFC 6830 (Proposed Standard), 2013.

[43] Ph. Flajolet, P. Zimmermann, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 1994.

[44] Robert W. Floyd. Algorithm 97 : Shortest path. *Commun. ACM*, 5(6) :345–, June 1962.

[45] Michael L. Fredman and Robert Endre Tarjan. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *J. ACM*, 34(3) :596–615, July 1987.

[46] V Fuller and T Li. RFC 4632 - Classless Inter-domain Routing (CIDR) : The Internet Address Assignment and Aggregation Plan. 2006.

[47] Lixin Gao. On inferring autonomous system relationships in the internet. *IEEE/ACM Trans. Netw.*, 9(6) :733–745, 2001.

[48] Lixin Gao and Jennifer Rexford. Stable internet routing without global coordination. In *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '00, pages 307–317, New York, NY, USA, 2000. ACM.

[49] D. Gardy and Y. Ponty. Weighted random generation of context-free languages : Analysis of collisions in random urn occupancy models. *GASCom*, 2010.

[50] Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[51] S. Gong and B. Jabbari. Optimal and Efficient End-to-End Path Computation in Multi-Layer Networks. In *ICC*, pages 5767–5771, 2008.

[52] Timothy Griffin and Gordon T. Wilfong. Analysis of the MED Oscillation Problem in BGP. In *ICNP*, pages 90–99. IEEE Computer Society, 2002.

[53] Tristan Groléat and Hélia Pouyllau. Distributed inter-domain SLA negotiation using Reinforcement Learning. In *Integrated Network Management*, 2011.

[54] D. Grossman. RFC3260 - New Terminology and Clarifications for Diffserv. RFC 3260 (Informational), 2002.

[55] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. RFC2597 - Assured Forwarding PHB Group. RFC 2597 (Proposed Standard), 1999.

[56] T. Hickey and J. Cohen. Uniform random generation of strings in a context-free language. *SIAM J. Comput.*, 12(4) :645–655, 1983.

[57] Tad Hogg and Lada Adamic. Enhancing reputation mechanisms via online social networks. In *Proceedings of the 5th ACM conference on Electronic commerce*, pages 236–237. ACM, 2004.

[58] J. E. Hopcroft, R. Motwani, and J. D. Ullman. From PDA's to Grammars. In *Introduction to Automata Theory, Languages, and Computation*, chapter 6.3.2, pages 247–251. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[59] C. Hornig. RFC894 - A Standard for the Transmission of IP Datagrams over Ethernet Networks, 1984.

[60] Yuval Filmus (http ://cstheory.stackexchange.com/users/40/yuval filmus) and Kaveh Ghasemloo (http ://cstheory.stackexchange.com/users/186/kaveh). Hardness of finding a word of length at most k accepted by a nondeterministic pushdown automaton. Theoretical Computer Science Stack Exchange. URL :http ://cstheory.stackexchange.com/q/4449 (version : 2011-01-21).

[61] ISO. Information processing systems – OSI reference model, international standards organization. Technical Report 7498, ISO, October 1984.

[62] Information Technology — Open Systems Interconnection — Basic Reference Model : The Basic Model. ISO/IEC 7498-1 :1994, ISO, November 1994.

[63] L. Pack Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement Learning : A Survey. *J. Artif. Intell. Res. (JAIR)*, 4 :237–285, 1996.

[64] Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3) :309–311, September 1978.

[65] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), 2005.

[66] Donald E. Knuth. A Generalization of Dijkstra's Algorithm. *Inf. Process. Lett.*, 6(1) :1–5, 1977.

[67] F. A. Kuipers and F. Dijkstra. Path selection in multi-layer networks. *Computer Communications*, 2009.

[68] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. In Shivkumar Kalyanaraman, Venkata N. Padmanabhan, K. K. Ramakrishnan, Rajeev Shorey, and Geoffrey M. Voelker, editors, *SIGCOMM*, pages 75–86. ACM, 2010.

[69] Mohamed Lamine Lamali(http ://cstheory.stackexchange.com/users/2954/lamine). Hardness of finding a word of length at most k accepted by a nondeterministic pushdown automaton. Theoretical Computer Science Stack Exchange. URL :http ://cstheory.stackexchange.com/q/4429 (version : 2011-01-21).

[70] Nicolas Le Sauze, Agostino Chiosi, Richard Douville, Hélia Pouyllau, Häkon Lonsethagen, Paola Fantini, Claudio Palasciano, Antonio Cimmino, MA Callejo Rodriguez, Olivier Dugeon, et al. ETICS : QoS-enabled interconnection for Future Internet services. *Future network and mobile summit*, 2010.

[71] Jens Liebehenschel. Lexicographical Generation of a Generalized Dyck Language. *SIAM J. Comput.*, 2003.

[72] M. L. Littman. Markov Games as a Framework for Multi-Agent Reinforcement Learning. In *ICML*, pages 157–163, 1994.

[73] Mary Madden and Aaron Smith. Reputation management and social media. 2010.

[74] George J Mailath and Larry Samuelson. Repeated games and reputations : long-run relationships. *OUP Catalogue*, 2006.

[75] H. G. Mairson. Generating Words in a Context-Free Language Uniformly at Random. *Inf. Process. Lett.*, 49(2) :95–99, 1994.

[76] A. Malis and M. Townsley. RFC 4623 - Pseudowire Emulation Edge-to-Edge (PWE3) Fragmentation and Reassembly, 2006.

[77] L. Martini, J. Jayakumar, M. Bocci, N. El-Aawar, J. Brayley, and G. Koleyni. Encapsulation Methods for Transport of Asynchronous Transfer Mode (ATM) over MPLS Networks. RFC 4717 (Proposed Standard), 2006.

[78] L. Martini, C. Kawa, and A. Malis. Encapsulation Methods for Transport of Frame Relay over Multiprotocol Label Switching (MPLS) Networks. RFC 4619 (Proposed Standard), 2006.

[79] L. Martini, C. Metz, T. Nadeau, M. Bocci, and M. Aissaoui. RFC 6073 -Segmented Pseudowire, 2011.

[80] L. Martini, E. Rosen, and N. El-Aawar. Encapsulation Methods for Transport of Layer 2 Frames over MPLS Networks. RFC 4905 (Historic), 2007.

[81] L. Martini, E. Rosen, N. El-Aawar, and G. Heron. RFC4448 - Encapsulation Methods for Transport of Ethernet over MPLS Networks, 2008.

[82] Piet Van Mieghem and Fernando A. Kuipers. Concepts of exact QoS routing algorithms. *IEEE/ACM Trans. Netw.*, 12(5) :851–864, 2004.

[83] Piet Van Mieghem, Hans De Neve, and Fernando A. Kuipers. Hop-by-hop quality of service routing. *Computer Networks*, 37(3/4) :407–423, 2001.

[84] John Moy. RFC 2328 - OSPF Version 2. Internet RFC 2328, 1998.

[85] J. Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2) :286–295, 1951.

[86] K. Nichols, S. Blake, F. Baker, and D. Black. RFC2474 - Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, 1998.

[87] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[88] W. Norton. DrPeering.net. `http://drpeering.net/`. Accédé le 29-4-2014.

[89] D. Oran. RFC 1142 - OSI IS-IS Intra-domain Routing Protocol. RFC 1142 (Informational), 1990.

[90] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*, volume 1 of *MIT Press Books*. The MIT Press, 1994.

[91] Ion Petre and Arto Salomaa. Algebraic Systems and Pushdown Automata. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, chapter 7, pages 257–290. Springer Publishing Company, Incorporated., 2009.

[92] J. Postel. RFC791 - Internet Protocol - DARPA Inernet Programm, Protocol Specification, 1981.

[93] J. Postel. RFC879 - The TCP Maximum Segment Size and Related Topics, 1983.

[94] M. L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[95] K. Ramakrishnan, S. Floyd, and D. Black. RFC3160 - The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), 2001.

[96] Omer F. Rana, Martijn Warnier, Thomas B. Quillinan, and Frances M. T. Brazier. Monitoring and reputation mechanisms for service level agreements. In *GECON*, 2008.

[97] Y. Rekhter, T. Li, and S. Hares. RFC 4271 - A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), 2006.

[98] Paul Resnick, Richard Zeckhauser, John Swanson, and Kate Lockwood. 'The value of reputation on eBay : A controlled experiment. *Experimental Economics*, 9(2) :79–101, 2006.

[99] M. Riegel. Requirements for Edge-to-Edge Emulation of Time Division Multiplexed (TDM) Circuits over Packet Switching Networks. RFC 4197 (Informational), 2005.

[100] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical report, Cambridge University Engineering Department, 1994.

[101] Jordi Sabater and Carles Sierra. Reputation and social network analysis in multi-agent systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems : part 1*, pages 475–482. ACM, 2002.

[102] Andreas Sackl, Sebastian Egger, Patrick Zwickl, and Peter Reichl. The QoE alchemy : Turning quality into money. Experiences with a refined methodology for the evaluation of willingness-to-pay for service quality. In *QoMEX*, pages 170–175, 2012.

[103] Andreas Sackl, Patrick Zwickl, and Peter Reichl. From quality of experience to willingness to pay for interconnection service quality. In *Proceedings of the 2012 International Conference on Networking*, IFIP'12, pages 89–96. Springer-Verlag, 2012.

[104] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27 :379–423, 623–656, 1948.

[105] I. Joseph Shapiro and Kumpati S. Narendra. Use of stochastic automata for parameter self-optimization with multimodal performance criteria. *IEEE Trans. Systems Science and Cybernetics*, 5(4) :352–360, 1969.

[106] Lloyd S Shapley. A value for n-person games. Technical report, DTIC Document, 1952.

[107] K. Shiomoto, D. Papadimitriou, JL. Le Roux, M. Vigoureux, and D. Brungard. RFC5212 - Requirements for GMPLS-based multi-region and multi-layer networks (MRN/MLN), 2008.

[108] S. P. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. *Machine Learning*, 38(3) :287–308, 2000.

[109] Kalika Suksomboon, Panita Pongpaibool, and Chaodit Aswakul. An equilibrium policy for providing end-to-end service level agreements in interdomain network. In *WCNC*, 2008.

[110] J. Touch. RFC6864 - Updated Specification of the IPv4 ID Field, 2013.

[111] J. Touch and M. Townsley. Tunnels in the internet architecture. http ://tools.ietf.org/html/draft-ietf-intarea-tunnels-00, 2010.

[112] Vytautas Valancius, Cristian Lumezanu, Nick Feamster, Ramesh Johari, and Vijay V. Vazirani. How many tiers ? : Pricing in the Internet Transit Market. In *SIGCOMM*, pages 194–205, 2011.

[113] Z. Wang and J. Crowcroft. Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE Journal on Selected Areas in Communications*, 14(7) :1228–1234, 1996.

[114] Stephen Warshall. A theorem on boolean matrices. *J. ACM*, 9(1) :11–12, January 1962.

[115] C. J.C.H. Watkins and P. Dayan. Technical Note : Q-Learning. In *Journal of Machine Learning Research*, pages 279–292, 1992.

[116] Dennis Weller and Bill Woodcock. Internet Traffic Exchange : Market Developments and Policy Challenges. OECD Digital Economy Papers 207, OECD Publishing, January 2013.

[117] Ming Xia, Massimo Tornatore, Charles U. Martel, and Biswanath Mukherjee. Risk-aware provisioning for optical WDM mesh networks. *IEEE/ACM Trans. Netw.*, 19(3) :921–931, 2011.

[118] Li Xiao, King-Shan Lui, Jun Wang, and Klara Nahrstedt. QoS extension to BGP. In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, pages 100–109. IEEE, 2002.

[119] X. Xiao, D. McPherson, and P. Pate. RFC 3916 - Requirements for Pseudo-Wire Emulation Edge-to-Edge (PWE3), 2004.

[120] W. Yao and B. Ramamurthy. A Link Bundled Auxiliary Graph Model for Constrained Dynamic Traffic Grooming in WDM Mesh Networks. *IEEE Journal on Selected Areas in Communications*, 23(8) :1542–1555, 2005.

[121] Yu Zhang and Mihaela van der Schaar. Reputation-based incentive protocols in crowdsourcing applications. In *INFOCOM, 2012 Proceedings IEEE*, pages 2140–2148. IEEE, 2012.

[122] H. Zhu, H. Zang, K. Zhu, and B. Mukherjee. A novel generic graph model for traffic grooming in heterogeneous WDM mesh networks. *IEEE/ACM Trans. Netw.*, 11(2) :285–299, 2003.

Résumé : Les applications multimédias et interactives occupent de plus en plus de place dans les réseaux et génèrent la plus grande partie du trafic. La plupart de ces applications nécessitent une garantie de paramètres (bande passante élevée, délai court, etc.) qu'on appelle Qualité de Service (Quality of Service - QoS). L'absence de QoS dans le réseau peut freiner le développement de ces applications. Déployer la QoS dans un réseau inter-domaine (i.e., réseau de réseaux) administré par des acteurs économiques différents est complexe : absence de coopération entre acteurs, hétérogénéité des technologies... Une solution possible est la définition de contrats bilatéraux entre domaines appelés Service Level Agreements (SLA). Ces contrats définissent un niveau de QoS qu'un domaine assure à un autre en échange de paiement. La mise bout à bout de ces SLA permet d'assurer la QoS en inter-domaine. Dans cette thèse, nous nous intéressons d'abord à l'étape de négociation de SLA : la sélection des SLA proposés par un domaine. Nous proposons des méthodes exactes et approchées (basées sur des algorithmes d'apprentissage) permettant aux domaines de proposer les SLA qui maximisent leurs revenus à long terme. Nous étudions également le cas où cette négociation est impactée par la réputation des domaines et nous adaptons notre solution pour prendre en compte ce paramètre. Au niveau de l'instanciation des SLA, nous nous intéressons au calcul de chemins qui prennent en compte les encapsulations et désencapsulations de protocoles (afin de pallier l'hétérogénéité technologique des domaines). En utilisant des outils de théorie des langages, nous proposons la première solution polynomiale au calcul de chemins dans un tel contexte. Nous étudions le problème contraint par la bande passante et proposons une solution générale au problème sous plusieurs contraintes de QoS.

Abstract : Multimedia and interactive applications play an increasingly important role in networks and are generating most of the network traffic. These applications generally require a guarantee of parameters (high bandwidth, short delay, etc.) called Quality of Service (QoS). The lack of QoS in networks can hinder the development of such applications. Deploying QoS in an inter-domain network (i.e., a network of networks managed by different economic players) is quite complex : lack of cooperation between the domains, technological heterogeneity... The introduction of bilateral contracts between domains, called Service Level Agreements (SLAs), is a possible solution. These contracts specify a QoS level that one domain provides to the other against remuneration. Combined, these SLAs ensure the QoS over inter-domain networks. In this thesis, we first focus on the SLA negotiation step, i.e., the selection of SLAs proposed by a domain. We provide exact and approximate methods (based on learning algorithms) that allow the domains to propose the SLAs that maximize their long term revenues. We also study the impact of the domain reputation on the SLA negotiation and we adapt our solutions to take into account this parameter. Regarding the SLA instantiation step, we focus on path computation processes that take into account the encapsulation and decapsulation of protocols (in order to overcome the domain technological heterogeneity). Using tools from Language Theory, we provide the first polynomial solution to the path computation problem in this context. We study the same problem under a bandwidth constraint and design a general solution to the problem under several QoS constraints.